NPS ARCHIVE
1997.12
DA SILVA FILHO, J.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

OBJECT RECOGNITION USING 2D SENSORS AND AUTONOMOUS VEHICLE NAVIGATION ISSUES

by

Jader Gomes da Silva Filho

December 1997

Co-Thesis Advisors: Yutaka J. Kanayama
Lynne L. Grewe
Gurnam S. Gill

THESIS
S49418

Approved for public release; Distribution is unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, Va 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December, 1997	Master's Thesis

4. TITLE AND SUBTITLE OBJECT RECOGNITION USING 2D SENSORS AND AUTONOMOUS VEHICLE NAVIGATION ISSUES

5. FUNDING NUMBERS

6. AUTHORS Filho, Jader G. S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT(maximum 200 words)

This research deals with the problem of extracting features from an image using wavelets and then using these features to recognize objects present in the image. This technique is applied to recognition of Unexploded Ordnance (UXO) objects. However, the concepts described here can be extended to recognition of other objects such as ships, missiles and aircrafts.
This work is performed as part of an ongoing effort to develop an autonomous vehicle capable of detecting UXOs.

14. SUBJECT TERMS Image Recognition, Unexploded Ordnance, Wavelets, Neural Networks, Motion Control

15. NUMBER OF PAGES 293

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

OBJECT RECOGNITION USING 2D SENSORS AND AUTONOMOUS VEHICLE NAVIGATION ISSUES

Jader Gomes da Silva Filho
Lieutenant, Brazilian Navy
B.S., Brazilian Naval Academy, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1997

NPS Archive
1997.12
Da Silva Filho, J.

Thesis
S49418
C.2

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

This research deals with the problem of extracting features from an image using wavelets and then using these features to recognize objects present in the image. This technique is applied to recognition of Unexploded Ordnance (UXO) objects. However, the concepts described here can be extended to recognition of other objects such as ships, missiles and aircrafts.

This work is performed as part of an ongoing effort to develop an autonomous vehicle capable of detecting UXOs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author thanks our Father in Heaven for all the bestowed blessings that made this work possible.

Also, thanks to Professor Yutaka Kanayama, Professor Lynne Grewe and Professor Gurnam Gill for their support and encouragement during all the research process.

Special thanks goes to the 787th EOD in Moffett Field, for allowing the collection of data used in this thesis.

I. INTRODUCTION

A. BACKGROUND

Modern warfare, as well as modern life, requires reliable and quick responding systems. To fulfill this need, autonomous systems capable of doing risky tasks have been a goal for decades. In practice, we are far from reaching this goal, but we have been slowly substituting machine systems for human operators in many activities, which has freed personnel from dangerous, boring and repetitive tasks.

Image recognition and localization is an important aspect of automation. It is a complex task with no general solution and has been used lately in many industrial and military applications. Recognition and localization are intimately related since we need to locate an object before being able to identify (recognize) it. In other words, we can not identify an object without knowing its location, but the reverse can be true.

In this thesis, wavelets and neural networks are used for object recognition. Features of objects are extracted in the wavelet domain and then a neural network is trained to recognize objects from these features.

One of the applications of the object recognition techniques is localization and/or identification of UXOs (Unexploded Ordnances). Detection of UXOs either during war or peace time is extremely dangerous. An autonomous system capable of substituting personnel in this task is highly desirable. This UXO recognition system can be applied to both battlefield strategic missions as well as cleanup of old battlefield sites. Figure 1 shows a typical UXO object. These objects are complex and hence challenging to recognize. The concepts presented in this thesis can also be used with other kind of sensor data such as magnetometer and infrared readings.

Figure 1. Typical UXO.

B. THESIS OUTLINE

The remaining chapters of this thesis deal with our subject matter in the following way. Chapter II gives a brief introduction to Neural Networks (NN) and the back-propagation learning method used to train them. In Chapter III, we discuss a classification system based on segmentation oriented geometric features and their use in a NN system for processing generic scenes. This chapter is to serve as an introduction to Object Recognition and to illustrate the usefulness of neural networks in detecting non-UXO objects. The geometric feature approach used in this chapter is based on the feature extraction algorithm in reference 1. Next, in Chapter IV, we discuss our UXO recognition system, as well as present experimental results. This system uses a wavelet based approach. This work will appear in reference 2 and is an extension of previous work described in reference 3. It capitalizes upon the multi-resolution nature of the wavelet domain to extract a set of features which is compact and yet sufficiently describes an object. Since this UXO detection system,

or a derivation of it, is intended to be integrated on Professor Kanayama's "Rotary" autonomous robot, we present modifications to the robot's motion control and navigation systems in Chapter V. Specifically, we discuss detection of edge landmarks, 3D modeling and neutral switching [Ref. 4, 5, 6]. Figure 2 shows a picture of the robot, also known by its nickname "Shepard." Finally, Chapter VI concludes this work.

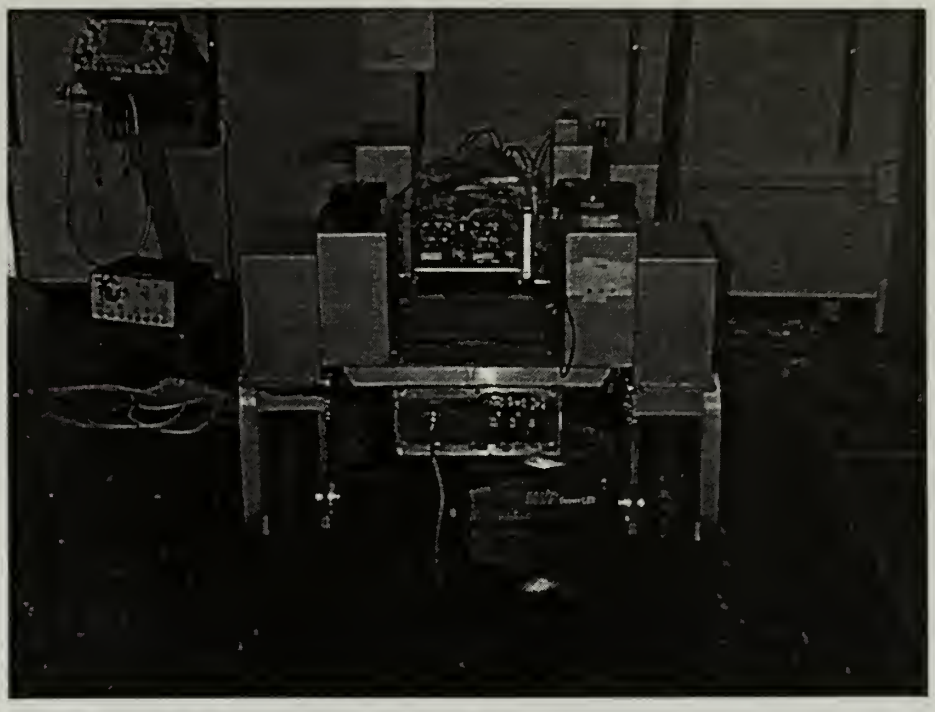


Figure 2. "Shepard"

II. NEURAL NETWORKS

A. INTRODUCTION

Generally the methods for pattern recognition may be classified as follows:

- Statistical pattern recognition techniques.
- Structural pattern recognition techniques.
- Hybrid techniques - Neural networks.

Neural networks approach has attracted considerable interest in recent years [Ref. 7]. It has been proved that a neural network can represent any continuous function with a defined degree of precision [Ref. 8]. As such, NN are ideal vehicles for signal representation and hence recognition.

Initially inspired by biological nervous system, a neural network consists of many interconnected neurons or processing elements (PEs) [Ref. 7], with similar characteristics, such as inputs, synaptic strengths, activations, outputs and biases. Each PE receives inputs from the preceding PEs and generates a scalar output. The first layer is called the input layer, the final layer is called the output layer and the middle layers are called hidden layers.

Figure 3 shows a basic model of a neuron [Ref. 9], where the non-linear function g, most commonly used, is the sigmoid:

$$g(x) = \frac{1}{1+e^{-x}} \ , \tag{II.1}$$

and the output is:

$$output = g\Big(\sum_{j=1}^{n} w_j input_j \ + \ w_B Bias\Big). \tag{II.2}$$

The "bias term" is used to fine tune the output of the node. In our research we do not use it, since the back-propagation training algorithm (given in the next section), finds the right weights for each node without the need of the bias term [Ref. 10].

Figure 4 presents a three-layer NN, where all nodes (represented by the circles), except the input nodes, have the same characteristics as the dashed circle in Figure 3. Note that the input nodes are just a pass-through for the input signal. This is the reason why, some authors do not count them when defining the number of layers in a NN. However, in this thesis, it will be to counted as a layer. In Figure 4, I_i is the input for input-node i, H_i is the output from hidden-node i, O_i is the output from output-node i (desired), w_{Iij} is the weight for the connection between input-node i and hidden-node j and lastly, w_{Hij} is the weight for the connection between hidden-node i and output-node j.

For a generic three-layer NN we have

$$H_i = g\Big(\sum_{j=1}^{n_I} w_{Iij} I_j\Big) \quad 1 \leq i \leq n_H, \tag{II.3}$$

$$O_i = g\Big(\sum_{j=1}^{n_H} w_{Hij} H_j\Big) \quad 1 \leq i \leq n_O. \tag{II.4}$$

Here n_I is the number of input nodes, n_H is the number of hidden nodes and n_O is the number of output nodes. In Figure 4, we have their values respectively equal to 3, 3 and 2.

B. TRAINING: BACK-PROPAGATION LEARNING ALGORITHM

Training a NN refers to presenting training data (consisting of input and corresponding output values) to the system so that it learns the best set of weights to produce, given the input data, the correct output. Once trained, a NN produces an output performing only a series of simple calculations as shown in Equation II.2. This feature makes NNs attractive. Back-propagation is a learning method developed in the mid-1980s by David Rumelhart [Ref. 11], and is based on finding the outputs at the last or output layer and calculating the errors or differences between the desired and the current outputs.

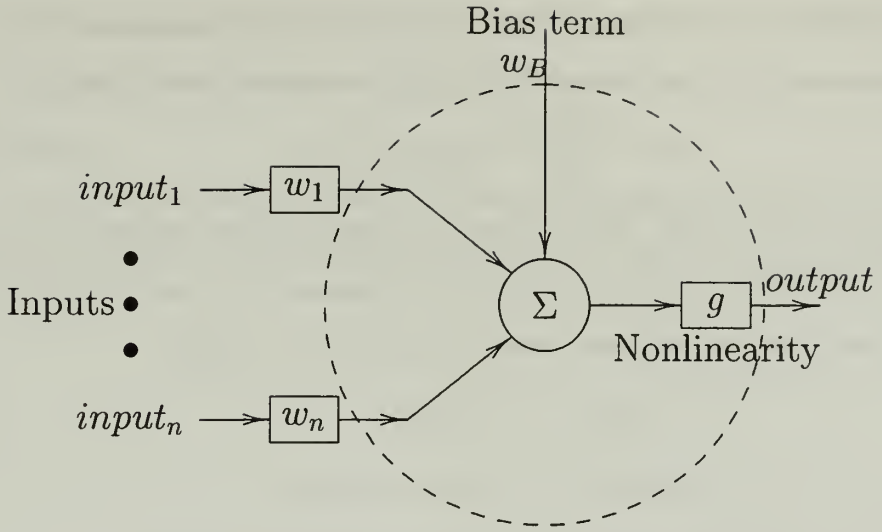


Figure 3. Basic Neuron (Node or PE) Model.

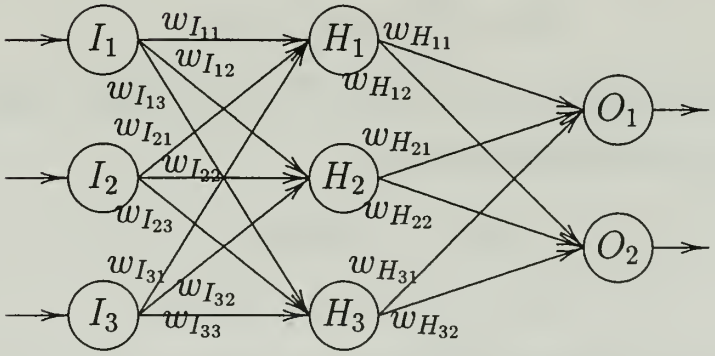


Figure 4. Three-Layer Neural Network.

These errors are then fed into the neural network in reverse order. This "backward" feeding of information from the output PEs to the input PEs is where back-propagation gets its name from. Thus, we must take a differential of the output and hence of $g(x)$. Because of the simple form of its differential, the sigmoid function is commonly used as $g(x)$. The differential when $g(x)$ equals to the sigmoid is:

$$g'(x) = \frac{dg(x)}{dx} = g(x)(1 - g(x)). \tag{II.5}$$

More specifically, what we do is make a change in the weights at each level as a function of the error and then (backward) propagate this error up the NN. The weights at the last layer are adjusted using the following formula:

$$\Delta w_{Hij} = \beta E_{Oj} H_i, \tag{II.6}$$

$$new\ w_{Hij} = w_{Hij} + \Delta w_{Hij}, \tag{II.7}$$

where β = learning rate $(0 < \beta \leq 1)$ and E_{Oj} is the given by,

$$E_{Oj} = O_j - O'_j, \tag{II.8}$$

where O'_j is the calculated output at the output-node j and O_j is the desired output.

For the hidden layer we need to calculate the error at this layer first. This is done, by calculating the "propagated error" from the output to the hidden nodes. The error for the hidden node i is given by,

$$E_{Hi} = \Big(\sum_{j=1}^{n_O} w_{ij} E_{Oj}\Big) g'(H_i). \tag{II.9}$$

Having computed the error at the hidden nodes, the updated weights of the input layer are given by the following equations:

$$\Delta w_{Iij} = \beta E_{Hj} I_i, \tag{II.10}$$

$$new\ w_{Iij} = w_{Iij} + \Delta w_{Iij}. \tag{II.11}$$

This process is done repeatedly until we have a small enough error for our output nodes. In our NN, the global error is calculated as

$$GE = \frac{\sqrt{\sum_{j=1}^{n_O} (O_i - O'_i)^2}}{n_O}. \tag{II.12}$$

In our studies, β is linearly varied between 0.1 and 0.0009. This variation will yield a faster convergence than a fixed β. [Ref. 9, 10, 12] are excellent resources for more details about NN.

We implemented a Lisp program, listed in Appendix A, to perform the training of a NN using back-propagation. All the NNs used in this thesis were trained with this program.

As will be discussed in the subsequent chapters, three-layer NNs, trained with the back-propagation learning algorithm, are used in our UXO detection system.

III. OBJECT RECOGNITION USING GEOMETRIC FEATURES

A. INTRODUCTION

A computer vision system [Ref. 1], that divides an image into regions with homogeneous characteristics is proposed for the purpose of classification and object recognition. This process is referred to as segmentation and extracts segments of the scene as features. The idea behind this technique is that these features represent the image information more compactly. The process of feature extraction is an important one in computer vision and is addressed in Chapter 4 with regards to UXO detection. This chapter focuses on one technique for feature extraction and illustrates that the feature-based object recognition system can be used for recognition of generic pictures.

1. Feature Extraction Algorithm

To divide a picture into regions of homogeneous characteristics, the system in [Ref. 1] goes through the following steps:

1. Image averaging of each four-cell square to compensate dithering.
2. Color gradient thresholding.
3. Clumping of the results using pixel regions.
4. Compute the 25 statistical property, listed in Table I.
    - Dimensions (statistics B-H and U) are measured in number of pixels.
    - Brightness (statistics K-Q and T) are computed with a 0-255 gray scale for each color.
    - Skews (statistics I and J) are proportional to the size of the box.
    - Diagonality (statistics V) is a fraction of the boundary length.
    - Curviness (statistics W) is in radians.
    - Correlations (statistics R and S) runs from -1 to 1.
    - Counts (statistics X and Y) are unadjusted.
5. Merge single-cell regions into their neighboring regions in a best-first approach. Regions properties were updated with each merge.

6. A final merging was then done utilizing a weighted sum of three factors (average color difference between the regions, the absolute difference in the neighbor-brightness variation over the regions and the weighted decrease in density of the bounding boxes) to rank the regions.

In Figure 6 we can see an example of the output from the feature extraction program [Ref. 1], for the input figure shown in Figure 5.

CODE	*Explanation*
A	region number
B	area in pixels
C	circumference in pixels
D	number of picture-boundary pixels
E	minimum x-coordinate
F	minimum y-coordinate
G	maximum x-coordinate
H	maximum y-coordinate
I	x-skew of center of mass from box center
J	y-skew of center of mass from box center
K	average red brightness
L	average green brightness
M	average blue brightness
N	standard deviation of red brightness
O	standard deviation of green brightness
P	standard deviation of blue brightness
Q	average brightness variation between adjacent cells
R	correlation of brightness with increasing x
S	correlation of brightness with increasing y
T	average strength of the region edge
U	smoothed-boundary length
V	smoothed-boundary diagonality
W	smoothed-boundary curviness
X	smoothed-boundary number of inflection points
Y	smoothed-boundary number of right angles
Z	whether boundary is opened or closed

Table I. The 26 Basic Region Statistics and Their Codes.

Figure 5. Original Picture.

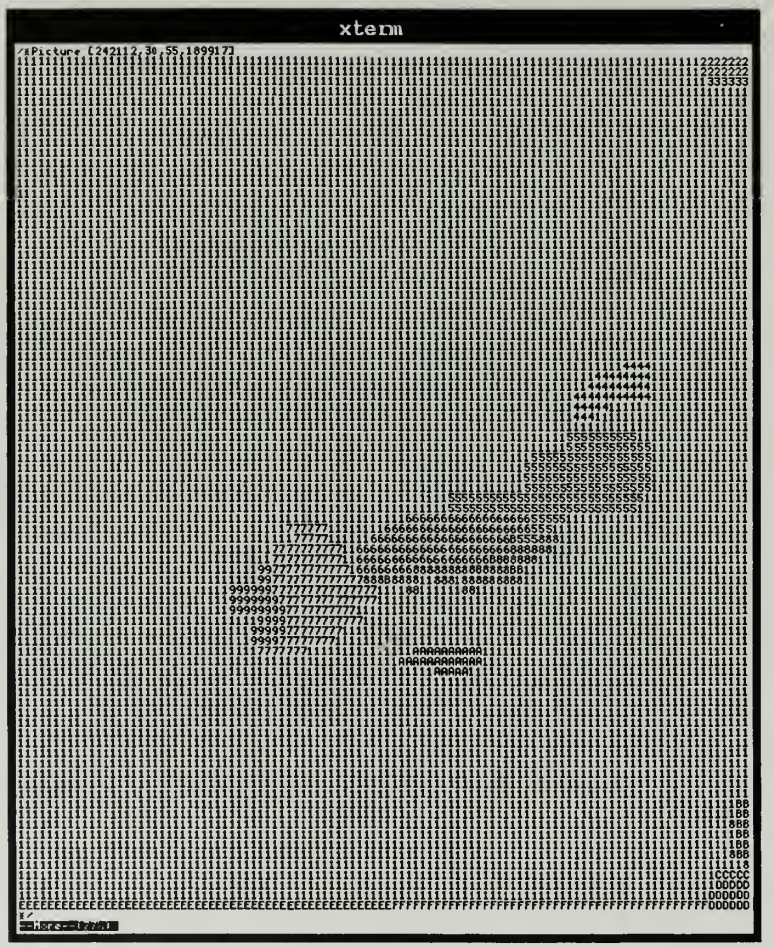


Figure 6. Output from the Feature Extraction Program.

2. Neural Network

The features extracted from the system in [Ref. 1] are used in a NN developed in Prolog (Appendix B), for recognition of segments (part of objects) in visual data. Again, this is to illustrate the potential usefulness of such a system for application to UXO detection. The UXO system is described in the next chapter. At first, we used 18 classes of possible objects, due to the difficulties in training, this number was dropped to 14 classes.

From the 26 elements in Table I 15 features were calculated using only 20 of the elements (Table II). The constant dividends observed in Table II were experimentally measured as the maximum observed values. Consequently the neural network had 15 input nodes, 15 hidden nodes and 4 output nodes. Fifty one training images were presented to the NN system. A prolog program reads the shapes from a file and gives the inputs to the neural net. A lisp implementation of back propagation was run and the best result for the weights gave a 33% of success rate when ran over the same set used to train the net.

The same prolog program, with the right weights, gives as output a list with as many terms as the number of inputs. Each element of the list contains the input number and a sequence of four numbers (0 and 1) that comprise a binary representing the class of the input, as shown in Table III.

After dividing the last three inputs (size dependent) by the area in pixels, the NN was able to be trained with a 100% of success rate over the training set.

Using the resulting weights, we were able to put the excluded classes (18 classes) back and still get good results (100% over the training set). The final classes used are listed in Table IV.

To test the system, we extracted features from Figure 7, resulting in the 23 regions presented in Figure 8 and listed in Appendix C. From these 23 regions, 11 were used to train the NN. Running the recognition system on the 12 remaining regions, it was able to identify 4 of them properly (33%).

INPUT	*DEFINITION*
1	$\frac{\sqrt{B}}{15}$
2	$\frac{D}{D+C}$
3	I
4	J
5	$\frac{K+L+M}{2000}$
6	$\frac{K}{650}$
7	$\frac{L}{650}$
8	$\frac{M}{650}$
9	$\frac{T}{500}$
10	$\frac{N+O+P}{50}$
11	$\frac{U}{B}$
12	$\frac{V}{B}$
13	$\frac{W}{B}$
14	$\frac{X}{B}$
15	$\frac{Y}{B}$

Table II. The 15 Input Values for the Neural Network.

CLASS	*BINARY*
Undefined	(0000)
Hair	(0001)
Building	(0010)
Leg	(0011)
Concrete floor	(0100)
Airplane	(0101)
Shadow	(0110)
Pole	(0111)
Sand	(1000)
Vegetation	(1001)
Pavement	(1100)
Cloud	(1101)
Sky	(1110)
Face	(1111)

Table III. The 14 Class Types.

B. RESULTS

To correctly identify different objects is a difficult problem. One problem of the vision system described in this chapter is that it is slow because the segmentation is a time consuming task [Ref. 1]. As it takes a long time to run, it is almost impossible to make experimental trials to find the best threshold, a parameter needed by the system, for each picture.

Normalizing the last five inputs by the area of the regions (in pixels) improved the results by a factor of 2.5. This basically produced a kind of scale invariance. The NN was able to find a correlation between same classes of different sizes and converge to a reasonable result.

Before those five inputs were normalized, the trained NN was able to successfully identify only 33% of the regions (classes) used in the training set. However, using the normalization described above, we recognized correctly 100% of the training set and 33% of the segments in a image not present in the training set. Even with the extreme complexity of the natural scene, we are able to obtain useful recognition

Figure 7. Original Picture.

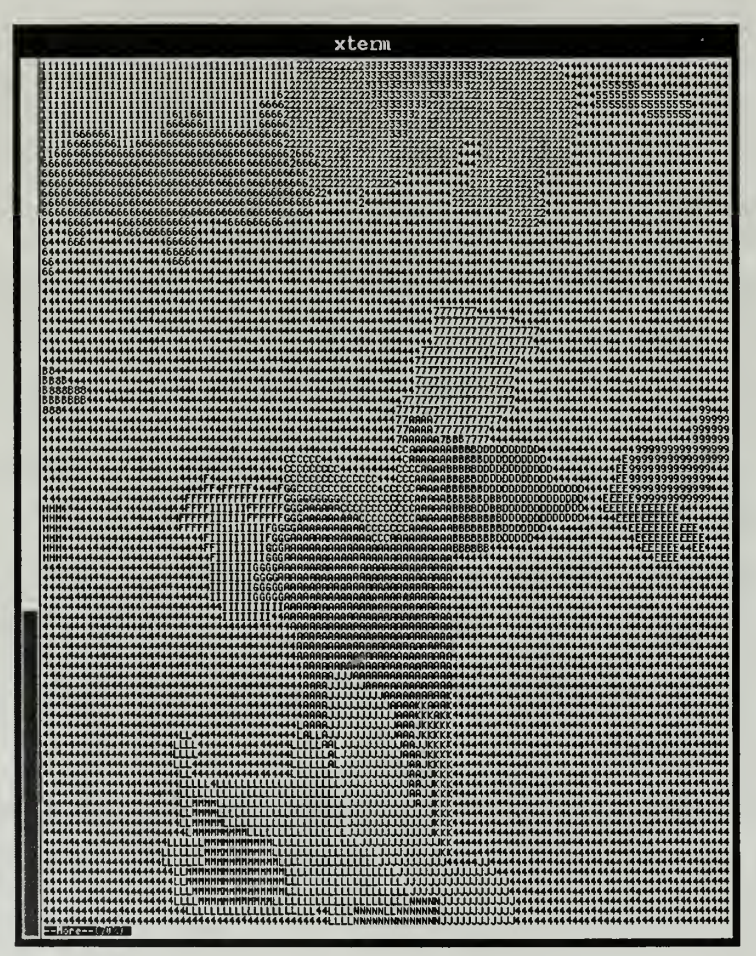


Figure 8. Output from the Feature Extraction Program.

CLASS	*BINARY*
Undefined	(10000)
Hair	(10001)
Building	(10010)
Leg	(10011)
Concrete floor	(10100)
Airplane	(10101)
Shadow	(10110)
Pole	(10111)
Sand	(11000)
Vegetation	(11001)
Hourse	(11010)
Body	(11011)
Pavement	(11100)
Cloud	(11101)
Sky	(11110)
Face	(11111)
Sky	(01110)
Face	(01111)

Table IV. The 18 Class Types.

results.

This suggests along with the vast body of research in vision that image object recognition may be useful in UXO detection. In the next chapter, we will describe our UXO detection system that utilizes computer vision techniques on 2D photometric input data.

IV. OBJECT RECOGNITION USING WAVELETS

A. INTRODUCTION

In this chapter image analysis techniques using wavelets are examined for use in the recognition of Unexploded Ordnances (UXOs). The system described here capitalizes upon the multi-dimensional and compact nature of the wavelet domain to perform the task of feature extraction. Subsequently, Neural Networks are used with training templates for recognition. The results obtained recommend the use of this system on a mobile platform for the task of UXO detection.

The application of UXO detection on a mobile platform allows us to assume that the objects presented to us on a fixed camera platform will fall in a small range of known scale. However, the objects may appear in any aspect, rotation and position in the scene.

The wavelet domain is highly suited to the recognition task due to its sparse yet descriptive qualities. Some of our previous research has indicated that in the task of localization, an optimal level of decomposition in the wavelet space can be chosen for an object [Ref. 3]. Thus, the complexity involved in recognition can be reduced by looking in this level of the multi-dimensional wavelet domain. Note that at this level, many of the key features are retained in the wavelet domain. The feature extraction phase starts by finding "special points of interest" at the chosen decomposition level and then calculating a set of features at each point including relative position, wavelet values, first and second moments. These interest points correspond roughly to areas of strong edge-type features. It is shown that these features are effective and reliable when used in a Neural Network system for recognition. An attribute of using these "points of interest" is that the information extracted for each point represents relatively local information about the object and when taken together with the other points gives a more global and hopefully distinctive description of the object under

question.

This system looks at using such local point features rather than the more global features of segments discussed in Chapter 3 because:

- It is faster to extract these feature.
- Exploitation of the wavelet domain, suggests the use of edge features rather than segments.
- Points are the simplest kind of features we can extract and are hence a good place to start when looking at the application of a feature based approach to object recognition.

A Neural Network is trained with features extracted from template training data. This data consists of samples for each UXO object and one or more template is needed for each aspect and rotation combination of the object. Experiments have indicated that small changes in scale should not affect results. A simple back propagation algorithm was employed for training the NN.

During the recognition phase, the scene is converted to the wavelet domain and at the pre-specified decomposition level the data is broken up into a series of templates. Each template is run through the Neural Network and tested for the existence of UXO objects. The results are promising and recommend further research as well as the possible use of this system on a mobile platform. It is important to note that the techniques described are not limited to image data but, could also be used with magnetometer and other sensor readings for UXO detection. This suggests one future avenue of research is that of combining different sensor data as input to the system.

B. WAVELETS

Superposition of functions to describe other functions has been used since Fourier's work in 1800's. In the mid-1980's, Stephane Mallat discovered relationships between pyramid algorithms and orthonormal wavelet functions which triggered the

current interest in wavelet transformations [Ref. 13]. Unlike Fourier transform, many modern wavelets have compact support (are zero outside a finite interval), which helps localize signals in time [Ref. 14].

1. Basic Mathematical Derivation

In this section, we describe the Daubechies wavelet transform [Ref. 14]. We start with the basic wavelet function $\psi(t)$ and its time translatable family

$$\psi_k(t) = \psi(t-k) \quad k \in \mathbf{Z}, \tag{IV.1}$$

where $\mathbf{Z}$ is the set of integers and k is the translation factor.

Next we introduce another parameter j to allow scaling of the basic function. Scale and time (or location) variations of the basic wavelet transform function are described by:

$$\psi_{j,k}(t) = 2^{\frac{j}{2}}\psi(2^j t - k) \quad k \in \mathbf{Z}. \tag{IV.2}$$

This bases functions are used to represent a function by expansion in the same way sines and cosines are used to represent a Fourier series expansion of that function. This is done by:

$$f(t) = \sum_{j,k} a_{j,k}\psi_{j,k}(t), \tag{IV.3}$$

where $a_{j,k}$ is the discrete wavelet transform values of $f(t)$. If $\psi_{j,k}(t)$ forms an orthonormal basis for the space of signals of interest, $a_{j,k}$ can be defined as the inner products

$$a_{j,k} = < \psi_{j,k}(t), f(t) > . \tag{IV.4}$$

One common way of calculating the wavelet $\psi(t)$ is to define first an orthonormal *scaling function* $\varphi(t)$. Following Equations (IV.1) and (IV.2), we have

$$\varphi_k(t) = \varphi(t-k) \quad k \in \mathbf{Z}, \tag{IV.5}$$

$$\varphi_{j,k}(t) = 2^{\frac{j}{2}}\varphi(2^j t - k) \quad k \in \mathbf{Z}. \tag{IV.6}$$

Using multi-resolution analysis [Ref. 15], we can write $\varphi(t)$ as

$$\varphi(t) = \sum_n h(n)\sqrt{2}\,\varphi(2t-k) \quad n \in \mathbf{Z}, \tag{IV.7}$$

where $h(n)$ is the scaling function coefficients and the $\sqrt{2}$ maintains the norm of the scaling function with a scale of two ($j = 1$).

Based on the orthogonality principle we also have

$$< \varphi_{j,k}(t), \psi_{j,l}(t) > = \int \varphi_{j,k}(t)\psi_{j,l}(t)dt = 0 \tag{IV.8}$$

for all appropriate $j, k, l \in \mathbf{Z}$. Now we can write $\psi(t)$ as

$$\psi(t) = \sum_n h_1(n)\sqrt{2}\,\varphi(2t-k) \quad n \in \mathbf{Z}, \tag{IV.9}$$

where $h_1(n)$ are the wavelet coefficients that help define the transform, and relates to the scaling function coefficients by

$$h_1(n) = (-1)^n h(1-n). \tag{IV.10}$$

Having defined $\psi(t)$ and $\varphi(t)$, we can now represent a function $g(t)$ as the series expansion

$$g(t) = \sum_{k=-\infty}^{\infty} c(k)\varphi_k(t) + \sum_{j=0}^{\infty}\sum_{k=-\infty}^{\infty} d(j,k)\psi_{j,k}(t), \tag{IV.11}$$

where the first summation gives a low resolution approximation of $g(t)$ and the second summation gives, for each increasing j, higher resolutions of the function. The coefficients $c(k)$ and $d(j,k)$ are defined as follows:

$$c(k) =< g(t), \varphi_k(t) > \;=\; \int g(t)\varphi_k(t)dt, \tag{IV.12}$$

$$d(j,k) =< g(t), \psi_{j,k}(t) > \;=\; \int g(t)\psi_{j,k}(t)dt. \tag{IV.13}$$

The following equations are necessary for calculating the scaling coefficients for the Daubechies wavelet [Ref. 16], which is the wavelet used in this thesis.

$$\sum_n h(n) \;=\; \sqrt{2} \tag{IV.14}$$

$$\sum_n h(n)h(n-2k) \;=\; \delta(k) = \begin{cases} 1 & \text{if k = 0} \\ 0 & \text{otherwise} \end{cases} \tag{IV.15}$$

$$\sum_n |h(n)|^2 \;=\; 1 \tag{IV.16}$$

For a length-4 coefficient sequence (Daubechies-4) we have the conditions:

$$h(0) + h(1) + h(2) + h(3) = \sqrt{2}, \tag{IV.17}$$

$$h(0)^2 + h(1)^2 + h(2)^2 + h(3)^2 = 1, \tag{IV.18}$$

$$h(0)h(2) + h(1)h(3) = 0. \tag{IV.19}$$

This yields a solution

$$h_{D4} = \left\{ \frac{1+\sqrt{3}}{4\sqrt{2}}, \frac{3+\sqrt{3}}{4\sqrt{2}}, \frac{3-\sqrt{3}}{4\sqrt{2}}, \frac{1-\sqrt{3}}{4\sqrt{2}} \right\} \tag{IV.20}$$

that is, the Daubechies-4 scaling coefficients.

2. Implementation

a. The DWT Algorithm

The algorithm used in this thesis was taken from reference 17, where the corresponding C code is given. The code was changed to work with C zero-based indexed arrays. Consider a 1D signal represented by a 1D column vector of length N. To get the wavelet Daubechies-4 transformation, which will be another 1D column vector of the same length, you will multiply the vector by the following $N \times N$ orthogonal matrix [Ref. 17]:

$$\begin{bmatrix}
h(0) & h(1) & h(2) & h(3) & & & & & & & & \\
h(3) & -h(2) & h(1) & -h(0) & & & & & & & & \\
 & & h(0) & h(1) & h(2) & h(3) & & & & & & \\
 & & h(3) & -h(2) & h(1) & -h(0) & & & & & & \\
\vdots & \vdots & & & & & \ddots & & & & & \\
 & & & & & & & h(0) & h(1) & h(2) & h(3) & \\
 & & & & & & & h(3) & -h(2) & h(1) & -h(0) & \\
h(2) & h(3) & & & & & & & & h(0) & h(1) & \\
h(1) & -h(0) & & & & & & & & h(3) & -h(2) &
\end{bmatrix}, \tag{IV.21}$$

where the blanks stand for zeroes. The values for $h(0), h(1), h(2)$ and $h(3)$ are given by Equation IV.20.

When this matrix is multiplied by the column vector, the result is another column vector of length N, where the odd rows represents smoothed values of adjacent rows and even rows represents differenced values of adjacent rows on the original column vector. The algorithm then groups all the smoothed rows in the first half of the column and all the differenced values on the second half. This process is recursively applied over the smoothed values, until we have just two remaining smoothed values on the top of the column vector. At each iteration i, the smoothed and differenced values correspond to 2^i adjacent rows of the original value.

For a two dimensional signal (like an image), the algorithm is initially applied over each row of the 2D matrix, and then over each column of the transformed matrix [Ref. 18]. The result is presented in Figure a, where D_i stands for the differenced operation over the i dimension of the matrix and analogously S_i stands for the smoothing operation. In our study, we just considered the portions that have been smoothed/differenced the same number of times in both directions (underlined diagonal elements in Figure a).

b. Image Manipulation

Our vision system uses the Matrox Image Library. This library provides several routines in C for manipulating images. In our work, we basically used the routines for reading, rotating, translating, scaling and displaying a picture. All the pictures fed into the DWT algorithm were cropped to be 512×512 pixels, in size.

During the coding phase, while extracting the features, we figure out that the DWT applied over the log of the greyscale values of the picture, results in a less noisy output, as illustrated in Figure 10. This was also used in [Ref. 19]. The feature points are more distinct in (b) than in (a).

$\underline{\ldots}$	$D_x \ldots$	$D_x S_y S_x \ldots$	$D_x S_y S_y S_y$
S_x $\ldots$	$\underline{D_x D_y}$ $\underline{S_x \ldots}$	$D_x S_y S_x S_y$	$D_x S_y S_y$
S_x S_x S_x $\ldots$	$S_x S_x$ $D_y S_y$	$\underline{D_x D_y S_x S_y}$	$D_x S_y$
S_x S_x S_x D_y	S_x $S_x D_y$	$S_x D_y$	$\underline{D_x D_y}$

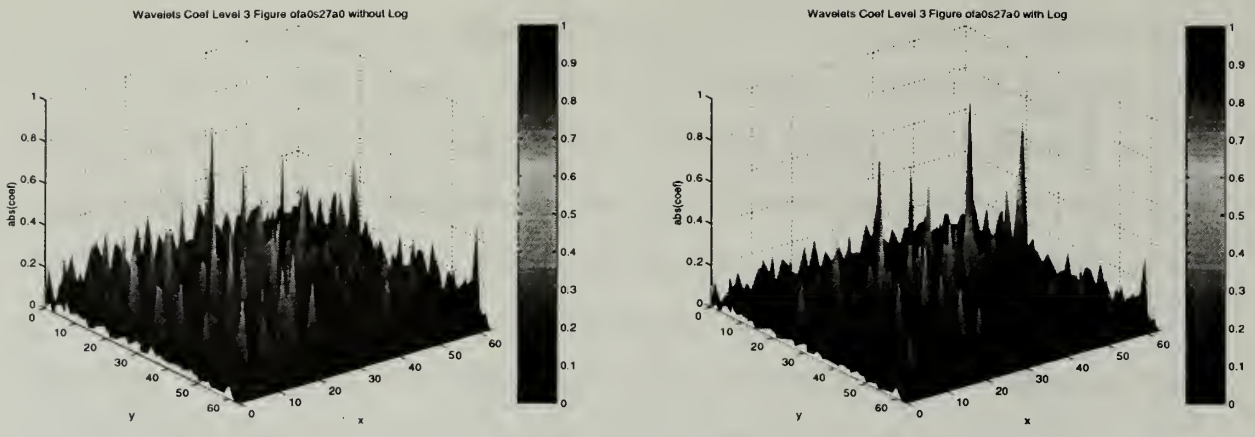
Figure 9. DWT Results over a 2D Matrix.

C. ASSUMPTIONS AND DATABASE

As discussed previously, the application of UXO detection on a mobile platform allows us to assume that the objects presented to us on a fixed camera platform will fall in a small range of known scale. Figure 2 illustrates the mobile platform "Shepard", that will soon receive a fixed camera assembly.

For these experiments, a 50mm lens was used and the range to the object was assumed to be approximately 6 feet. The data was collected with a camera manually to mimic the robotic assembly. This was necessary as the UXO objects were not available on site and travel to Moffett Field was required. A digital camera was used for collecting some of the training and some of the test data, the other was taken with a CCD analog camera. We assumed also that the camera creates 512×512 pixel images. It is important to note that any different camera configuration can be accommodated, given that new training data is provided.

Note that the objects may appear in any aspect, rotation and position in the scene. Figures 11 ,12 and 13 show three UXOs that are used in our database. Possible



(a) Without Log (b) With Log

Figure 10. The Feature Points Are More Distinctive in the Wavelet of the Log Domain Shown in (b) Compared to Those Extracted for the Same Scene in the normal Wavelet Domain (a)

aspects of each object is shown. We define an aspect as a unique view of an object such that different surfaces of the object are viewable in each aspect. For this study, a single aspect of each object was used (aspect 0) as a proof of concept.

D. FEATURE EXTRACTION

1. "Interest Point" Detection

There are many kinds of features that can be extracted as was discussed previously, in our generic image recognition system of Chapter 3. There exists a trade-off between how local and global a feature is and typically the time and ease of extraction of the feature. While global features are desirable, it is often not possible to define nor extract such features reliably. Global features such as surfaces work best with very simple objects like planar objects and do not work for fluidly shaped objects. While UXOs are man-made objects they are very complex and have smoothly varying shapes as can be seen in Figures 11, 12 and 13. This combined with the fact that

(a) Aspect 0

(b) Aspect 1

(c) Aspect 2

(d) Aspect 3

(e) Aspect 4

Figure 11. 81mm Mortar (class 1).

(a) Aspect 0

(b) Aspect 1

(c) Aspect 2

(d) Aspect 3

(e) Aspect 4

Figure 12. 105mm HEAT Round (class 2)

(a) Aspect 0

(b) Aspect 1

(c) Aspect 2

(d) Aspect 3

(e) Aspect 4

Figure 13. 105mm Artillary Round (class 3)

many of the objects look similar, was why we choose to look for more local rather than global features.

A type of local feature successfully used in computer vision systems is the edge. An edge measures the local discontinuity in greyscale values. The larger the discontinuity, the greater the edge strength. In our system, we detect "Interest Points" in a wavelet domain that can be loosely thought of as edge points of greater strength. Another reason why the wavelet transform is used is because its multi-resolution nature can lead to improved computational performance if lower-resolution levels can be examined.

After converting the image to the wavelet domain, we have chosen to perform feature extraction and subsequent recognition at the 3rd level of decomposition. This means that instead of examining the entire 512×512 image we are now looking only at a 64×64 image. This level was chosen using the results described in [Ref. 3] that discusses how to pick an optimal level in which to perform object localization. Figure 14, shows the 3rd decomposition level for the image in Figure 12 (b). While Figure 15 show the whole DWT output of that image. Notice that it still retains much of the detail needed to identify the object.

The "Interest Points" in this level of the wavelet domain are extracted as the top ten wavelet pixel values. These points will correspond to the strongest edge points at this level of resolution. As discussed in the next section, we use not only the location of each point but, also extract other attributes to create a feature vector for each "Interest Point". Our hypothesis is that these "Interest Point" feature vectors while containing local information can together be used to create a description that will distinguish different objects in particular aspects and angles in the UXO database from each other.

After a point is selected and its feature vector calculated, the wavelet values in the surrounding 3×3 neighborhood are set to zero to avoid their selection as "Interest Points". This will guarantee a distribution of points in the wavelet domain. Also we

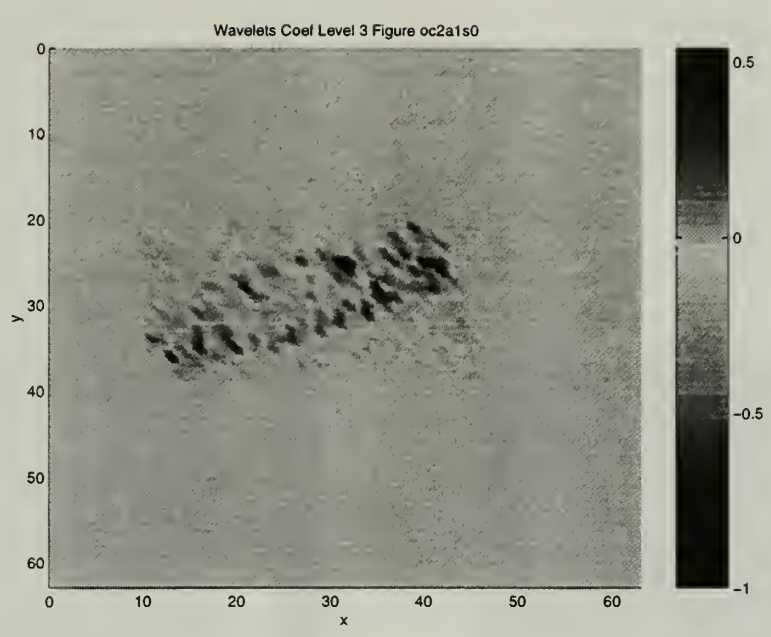


Figure 14. Third Level Wavelet Decomposition.

sort the "Interest Points" and their feature vectors by their distances to the origin of the wavelet domain (upper-left hand corner).

Figure 16 shows for an aspect of each UXO object the "Interest Points" detected. In Figure 17, objects are shown at different angles but, the same aspect. Notice that the "Interest Points" are different for different aspects and angles. In Figure 18 shows an object at different locations in the image and the fact that the location of the "Interest Points" are close to constant.

2. Features at each "Interest Point"

Each "Interest Point" extracted is described by a feature vector. This vector consists of the location of the point, its wavelet value, the average wavelet value in a local neighborhood and the variance of the wavelet values in a local neighborhood. For the purpose of these experiments, a neighborhood of 3×3 was chosen.

Note that the location of each point is in reference to the centroid of all of the detected "Interest Points" (Equations IV.22 and IV.23). This is important as we do not want the absolute location of the "Interest Points" but, their locations relative

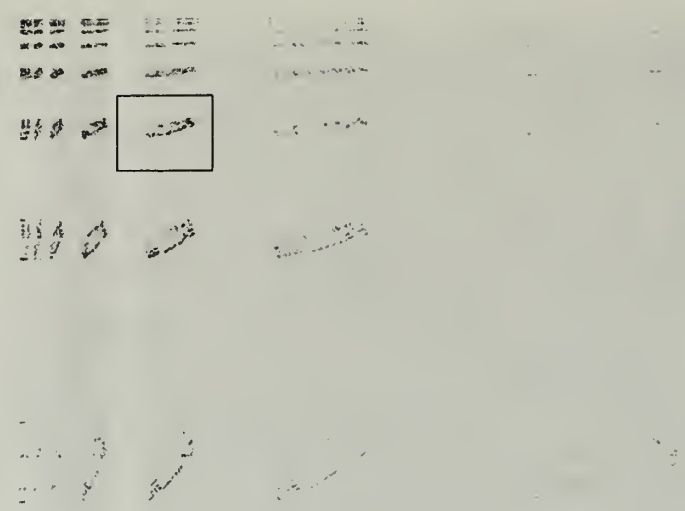


Figure 15. Wavelet Decomposition.

to each other. The wavelet values are first divided by their maximum absolute value to provide normalization. This will be useful later during the template scanning, to assure that the same range of values will be fed into the NN. Note that as mentioned before, every template has its value normalized before the feature vector is extracted. This, together with the relative location of the features, guarantees that the range of the values for the scanned feature vector will match the one used during training. The 9 values of a 3×3 neighborhood of a selected "Interest Point", will be combined according to Equations IV.24 and IV.25 to give the feature vector's mean and variance, respectively.

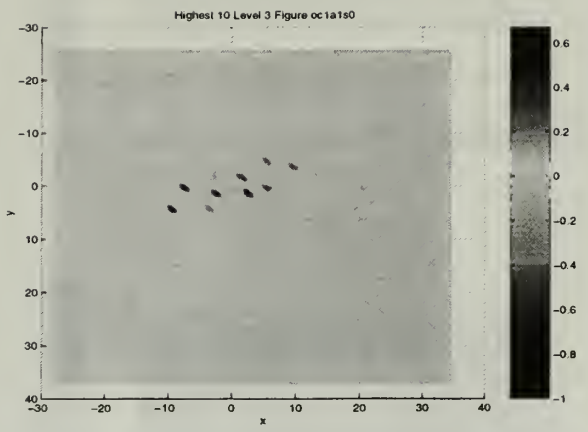
$$\overline{x} = \frac{\sum_{i=1}^{9} x_i}{9} \tag{IV.22}$$

$$\overline{y} = \frac{\sum_{i=1}^{9} y_i}{9} \tag{IV.23}$$

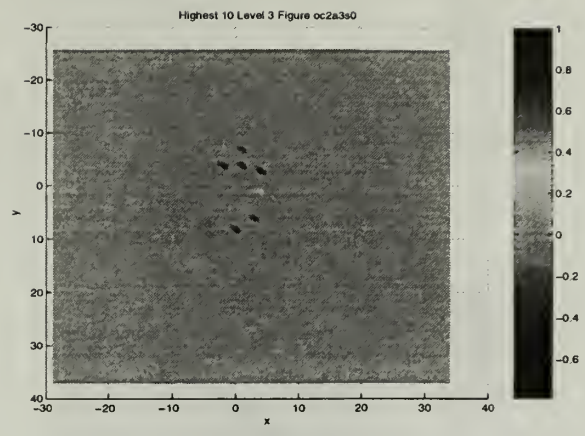
$$\mu = \frac{\sum_{i=1}^{9} Value_i}{9} \tag{IV.24}$$

$$\sigma^2 = \frac{\sum_{i=1}^{9} Value_i^2 - \mu^2}{9} \tag{IV.25}$$

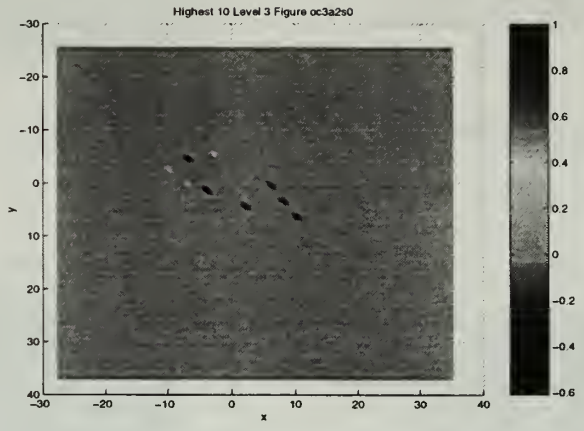
(IV.26)



(a) Feature Points for Figures 11 (b)

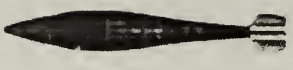


(b) Feature Points for Figures 12 (d)

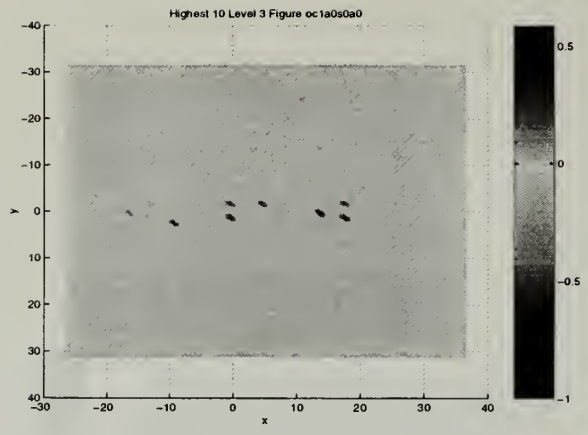


(c) Feature Points for Figures 13 (c)

Figure 16. Feature Points Detected for Different Aspects. Darker Points Represent Larger Wavelet Values.

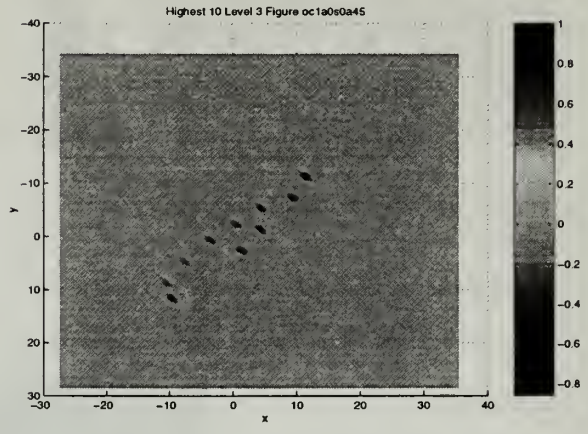


(a) Object Class 1 Aspect 0 at 0°



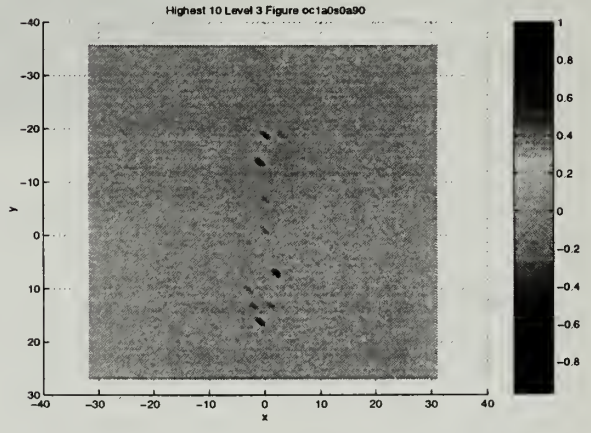
(b) Feature Points Extracted from (a)

(c) Object Class 1 Aspect 0 at 45°



(d) Feature Points Extracted from (c)

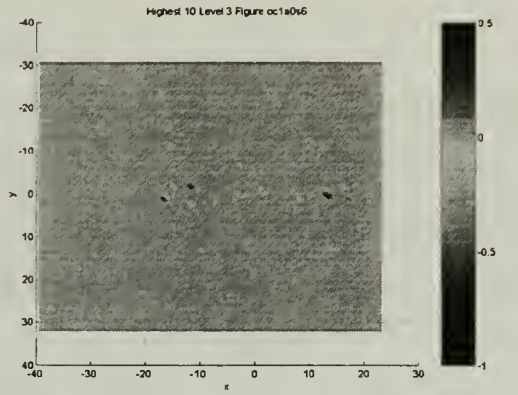
(e) Object Class 1 Aspect 0 at 90°



(f) Feature Points Extracted from (e)

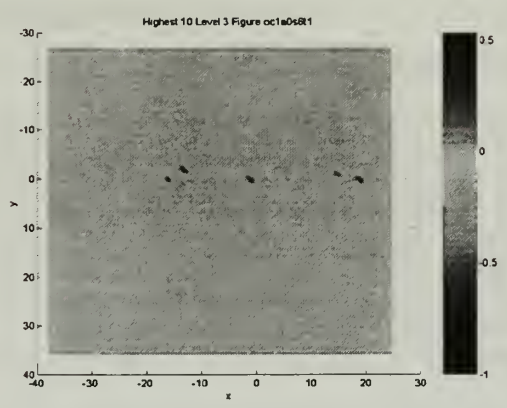
Figure 17. Feature Points Detected for Different Angles

(a) Object Class 1 Aspect 0 No Translation



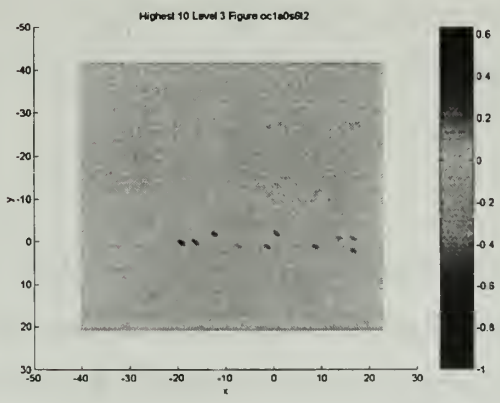
(b) Feature Points Extracted from (a)

(c) Object Class 1 Aspect 0 Translated



(d) Feature Points Extracted from (c)

(e) Object Class 1 Aspect 0 Translated



(f) Feature Points Extracted from (e)

Figure 18. Feature Points Detected for Different Translations

Feature Vector	X	Y	Value	Mean	Variance
1	-7.400	0.300	0.673	0.255	0.045
2	-2.400	-1.700	-0.230	0.091	0.005
3	-9.400	4.300	-0.731	0.262	0.044
4	-2.400	1.300	0.526	0.121	0.025
5	1.600	-1.700	0.431	0.158	0.017
6	-3.400	4.300	-0.404	0.091	0.015
7	5.600	-4.700	0.285	0.094	0.008
8	2.600	1.300	-1.000	0.270	0.099
9	5.600	0.300	0.336	0.198	0.033
10	9.600	-3.700	-0.522	0.173	0.022

Table V. Feature Vectors for Figure 16 (a)

Table V lists the feature vectors extracted for the UXO shown in Figure 16 (a). Table VI lists the feature vectors extracted for a different UXO object shown in Figure 16 (b). Notice that the feature vectors are different enough to distinguish the two apart.

Feature Vector	X	Y	Value	Mean	Variance
1	-2.000	-6.900	0.390	0.176	0.032
2	-2.000	-3.900	1.000	0.361	0.133
3	1.000	-6.900	0.848	0.306	0.074
4	-5.000	2.100	0.457	0.094	0.024
5	1.000	-3.900	0.805	0.253	0.077
6	4.000	-2.900	-0.791	0.314	0.080
7	-4.000	7.100	0.507	0.198	0.032
8	4.000	1.100	0.337	0.167	0.010
9	0.000	8.100	-0.450	0.146	0.017
10	3.000	6.100	-0.314	0.130	0.009

Table VI. Feature Vectors for Figure 16 (b)

3. Training Data: Examples

Figure 17 shows some of the training data used. Unfortunately, we were only able to obtain 27 images per aspect of the UXO objects. Additionally, for ease of the test study, we chose a single aspect of each object. Rotational versions of each aspect where generated from these samples by rotating the images at increments of 0, 45 and 90 degrees.

E. RECOGNITION SYSTEM

In this section, the structure of the Recognition System is described. It consists of using Neural Networks to implement a template matching scheme.

While we have made the hypothesis that the features extracted at each of the ten interest points for an object will adequately describe an object, it is not true that an image consisting of possibly multiple objects in a more natural setting than the training data will have the top ten interest points only belonging to a single object. Thus, we must somehow look at multiple combinations of ten interest points in the scene data, each representing a possible object hypothesis. Given this is the case, a scheme involving template scanning was created to examine only portions of the scene at a time. In each of these template portions, the top ten interest points are fed into our Neural Network Processing System for the purpose of verification of the hypothesis that they correspond to the interest points of a particular object at a particular angle and aspect combination. Note that the center of the template itself yields the location of the object in the scene.

1. Template Scanning

Because we have objects of varying size and the largest object spans approximately only 75% of the image area in its longest aspect, we do not want to consider the entire image but, only regions of the image as potentially containing objects. By looking in smaller regions, there is less likelihood that we will detect "Interest Points" from the image's background. Hence we perform a template scanning procedure to

divide the image under consideration into smaller regions.

Recall, that the system is actually examining the 3rd decomposition level of the wavelet domain which in this case is 64x64 pixels in size. Consequently, we examine different regions of the image that are 64×16, 16×64, 48×16, 16×48, 38×16, 16×38, 36×28 and 34×27, 45×36 in size. Other sized templates could be used as a way of improving system performance. Choice of these template sizes should be a function of the range of the various aspect shapes in the database. The first 6 templates where chosen to try to capture the near horizontal and near vertical placement of objects in the scene. The last three templates attempt to capture diagonal rotations of objects in the scene. Again all of these sizes where manually chosen and the determination of these sizes is a possible future area of research.

Before performing the template scanning, we set the 10% lowest wavelet values (3rd decomposition) to zero. In doing this, we are removing the Gaussian noise from the picture [Ref. 20].

There are multiple instances of the smaller sized templates and some "template scanning" scheme is needed to extract them. Below is the algorithm used:

```
for(row = 0; row + window_vertical_size <= 64; row+=SCAN_STEP)
  for(column = 0; column + window_horizontal_size <= 64;
      column+=SCAN_STEP)
    find_scan_ten_highest(row,column,window_vertical_size,
                          window_horizontal_size);
```

The feature vectors from each candidate template are placed in a file in the order of their extraction for processing by the Neural Network System described next.

2. Neural Network System

In this section, we describe the Neural Network System that is used to perform recognition of the UXO objects given the input of the ten feature vectors. First, the structure of the Network System is described. Next, the training process is described. For a review of Neural Networks and a discussion of learning methods used see the previous chapter on Neural Networks.

a. Structure

Figures 19 and 20 are diagrams of the Neural Network Systems that were developed to perform recognition given our 10 feature vectors as input. We tested the system using a single NN for all classes as well as a NN for each separate class (Multiple NN). In the multiple NN case, we will refer to a Neural Network for Object i as NNi. Neural Network NNi is responsible for determining whether or not the input feature vectors classify as feature vectors from Object i.

1. Multiple NN Case

   The output of each NN consists of: the most likely class, as well as a measure of confidence. Notice that the outputs from the NNs in Figure 20 are passed to a Voting algorithm, that will determine which if any of the NNi's indicate the presence of an object and if so the identity, aspect and angle of it. If the error is small enough (measure of confidence is great enough), than the location of the template is marked as containing the detected object. As many templates are processed, it is possible to find multiple objects located in the scene. The internal structure of each NN is identical and depicted in Figure 21. Notice that there are 50 inputs to feed in the 5 elements of the 10 "Interest point's" feature vectors. The middle layer contains 50 nodes. The output consists of N nodes representing the N classes.

2. Single NN Case

   We also created two versions of the single NN architecture. The first case consisted of three output nodes representing the three objects at aspect 0 and angle 0. The second case consisted of nine output nodes representing the following classes:

- {obj1, 0°, aspect0} {obj1, 45°, aspect0} {obj1, 90°, aspect0}
- {obj2, 0°, aspect0} {obj2, 45°, aspect0} {obj2, 90°, aspect0}
- {obj3, 0°, aspect0} {obj3, 45°, aspect0} {obj3, 90°, aspect0}

These classes do not represent but are only a example of all the possible objects, aspects and angles combinations. However, we believe it is a sufficient enough sample to test the usefulness of this recognition system on UXO detection.

The output of each NN during testing however will typically not be binary. Because of the sigmoid function applied as described in Chapter 2, the output of each node varies from 0 to 1. For each node, we then round its output to either 0 or 1 (threshold of 0.5). The difference between the actual output vector and the binary rounded vector is used as the error measure. Note that a measure of confidence can be taken to be inversely related to this error measurement. For example, in the 3 class case, suppose we have

$$Output = [0.3\ 0.2\ 0.9] \tag{IV.27}$$

rounding we have:

$$Rounded\ Output = [0\ 0\ 1] \tag{IV.28}$$

Thus, the decision is:

$$Decision = Class\ 1\ with\ error = \frac{\sqrt{0.3^2 + 0.2^2 + 0.1^2}}{3} = 0.1247 \tag{IV.29}$$

b. Presentation of Results

Here we discuss results based on the order of sets of consecutive templates with similarly low error values. The following results are printed-out to the screen for user inspection:

```
A8: Class identity of 8 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  8-consecutive
    sets.
A7: Class identity of 7 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  7-consecutive
    sets.
A6: Class identity of 6 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  6-consecutive
    sets.
A5: Class identity of 5 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  5-consecutive
    sets.
A4: Class identity of 4 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  4-consecutive
```

```
    sets.
A3: Class identity of 3 consecutive templates yielding same identity
    and similar error. This is the lowest of all such  3-consecutive
    sets.
```

By similar error we mean that the values fall within Δ of each other. For this thesis $\Delta = 10^{-8}$. Ideally, you would choose the answer A_i where you maximize i at the same time as minimizing the corresponding error. This was implemented by using the equation

$$V_i = \frac{i}{8} + \frac{E_{max}}{E_i}, \tag{IV.30}$$

and picking the consecutive windows which maximizes its value, where $E_{max} = \max E_1, E_2, ..., E_8$ and E_i is the corresponding error. In practice, picking A_i with the least error or minimizing V_i, gave very similar results.

c. Training

Training takes place using the back-propagation learning method described in Chapter 2. A set of 27 images for every aspect & angle combination for an object is presented as input to the NN along with the correct classification information. In the both NN cases (multiple and single NNs), due to the limited number of training samples available, and a need to also detect objects that are not Object i, a set of training images of other objects besides Object i are presented as input to NNi. These examples work as counter examples for the training process.

The network is trained until it converges to an error less than .02 (for three-class case) or .05 (for nine-class case).

The approximate time it took to train each Neural Network ranged from 10 hours to 7 days on a SGI O2 machine with a 180 MHz R5000 processor.

F. IMPLEMENTATION

The Feature Extraction Program and the Scanning Program (Appendix D) are implemented in Microsoft Visual C++. The Neural Network Training Program

(Appendix A) as well as the Neural Network Recognition Program (Appendix E) are implemented in Allegro Common Lisp 4.2 on an SGI machine.

All the details about the implementation of the recognition system presented in this chapter, are listed in Appendix J.

G. RESULTS

1. Test Data

We have grouped the Testing data into four successfully more difficult sets of scene data. The first set consist of samples of the training data and the entire image is presented as a template, rather than performing template scanning. The second set consists again of training data where we do perform template scanning. The third set consists of objects against a background. The fourth set depicts multiple objects in a scene.

2. Accuracy

a. Three-Class Case

All of images in Set #1 were classified correctly. Table VII shows the results for Test Set #2 and Table VIII shows the results for Test Set #3, for the multiple NN. While Table IX shows the results for Test Set #2 and Table X shows the results for Test Set #3, for a single NN. Table XI shows the results for set #4.

For the multiple NN case we have (Min Error):

- Out of the 81 samples in Test Set #2, 74 (91.4%) are classified correctly by object identification, 79 (97.5%) correctly located and 74 (91.4%) are correctly located and identified.
- Out of the 21 samples in Test Set #3, 13 (61.9%) are classified correctly by object identification , 15 (71.4%) correctly located and 13 (61.9%) are correctly located and identified.

For the single NN case we have (Min Error):

- Out of the 81 samples in Test Set #2, 74 (91.4%) are classified correctly by object identification, 78 (96.4%) correctly located and 74 (91.4%) are correctly located and identified.

Set2 (27 Pictures)	Correctly Identified	Correctly Located	Correctly Identified and Located
Min Error	c1 = 26 c2 = 24 c3 = 24	c1 = 26 c2 = 27 c3 = 26	c1 = 26 c2 = 24 c3 = 24
Formula	c1 = 26 c2 = 24 c3 = 24	c1 = 26 c2 = 27 c3 = 26	c1 = 26 c2 = 24 c3 = 24
Exist in 3 or more consecutive windows	c1 = 26 c2 = 25 c3 = 26	c1 = 26 c2 = 27 c3 = 26	c1 = 26 c2 = 25 c3 = 26

Table VII. Results for Multiple NN 3 Classes Set 2

- Out of the 21 samples in Test Set #3, 11 (52.4%) are classified correctly by object identification , 16 (76.2%) correctly located and 11 (52.4%) are correctly located and identified.
- Out of the 6 samples in Test Set #4, 4 (66.7%) are classified correctly by object identification , 5 (83.3%) correctly located and 4 (66.7%) are correctly located and identified.

In Figure 22 shows an example of the results obtained for the single NN case with a set #3 picture. The box in Figure 22 (b) represents the location of the best answer found.

b. Nine-Class Case

All of the images in Set #1 are classified correctly. Tables XII, XIII and XIV show the results for Test Set #2 and Tables XV, XVI and XVII show the results for Test Set #3, for a single NN. Tables XVIII, XIX and XX show the results for set #4.

For the single NN case we have (Min Error):

- Out of the 243 samples in Test Set #2, 184 (75.7%) are classified correctly by object identification, 205 (84.4%) with correct angle information, 225 (92.6%)

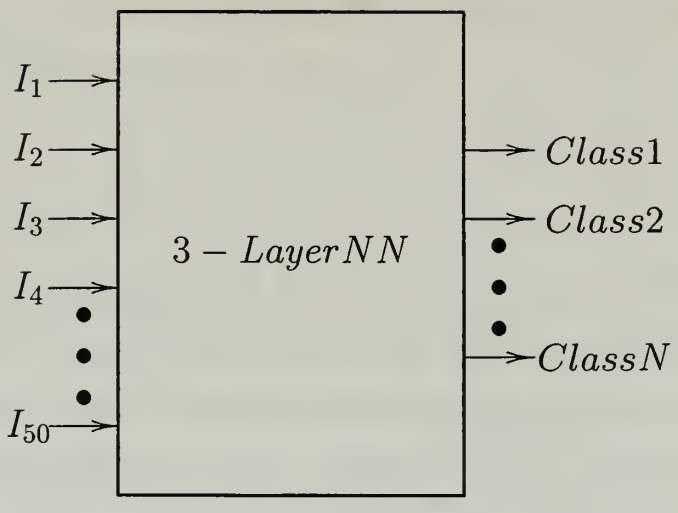


Figure 19. Diagram for a Single Neural Network.

Set3 (7 Pictures)	Correctly Identified	Correctly Located	Correctly Identified and Located
Min Error	c1 = 4 c2 = 6 c3 = 3	c1 = 4 c2 = 7 c3 = 4	c1 = 4 c2 = 6 c3 = 3
Formula	c1 = 4 c2 = 6 c3 = 3	c1 = 4 c2 = 7 c3 = 4	c1 = 4 c2 = 6 c3 = 3
Exist in 3 or more consecutive windows	c1 = 4 c2 = 6 c3 = 5	c1 = 4 c2 = 7 c3 = 5	c1 = 4 c2 = 6 c3 = 4

Table VIII. Results for Multiple NN 3 Classes Set 3.

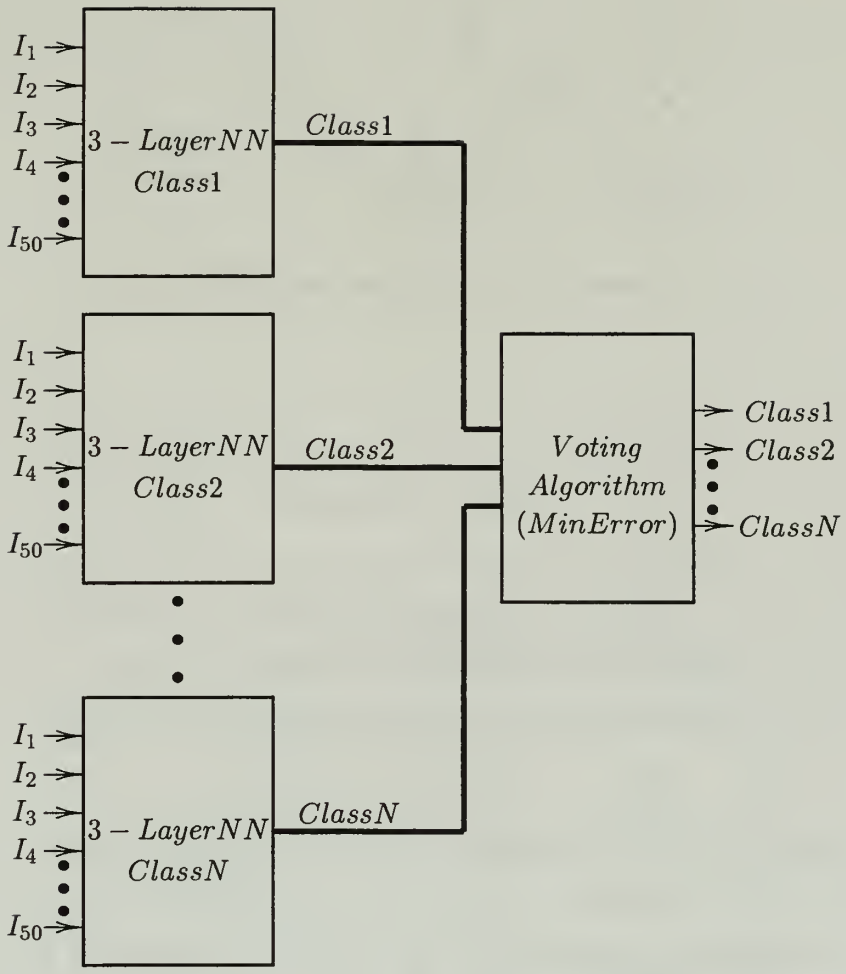


Figure 20. Diagram for Multiple Neural Networks.

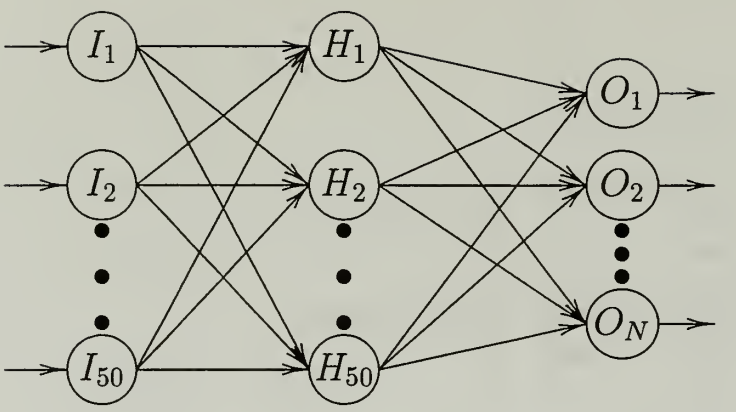


Figure 21. Three-Layer Neural Network.

Set2 (27 Pictures)	Correctly Identified	Correctly Located	Correctly Identified and Located
Min Error	c1 = 25 c2 = 24 c3 = 25	c1 = 25 c2 = 27 c3 = 26	c1 = 25 c2 = 24 c3 = 25
Formula	c1 = 25 c2 = 24 c3 = 25	c1 = 25 c2 = 27 c3 = 26	c1 = 25 c2 = 24 c3 = 25
Exist in 3 or more consecutive windows	c1 = 25 c2 = 26 c3 = 25	c1 = 25 c2 = 27 c3 = 26	c1 = 25 c2 = 26 c3 = 25

Table IX. Results for a Single NN 3 Classes Set 2

Set3 (7 Pictures)	Correctly Identified	Correctly Located	Correctly Identified and Located
Min Error	c1 = 5 c2 = 3 c3 = 3	c1 = 5 c2 = 7 c3 = 4	c1 = 4 c2 = 4 c3 = 3
Formula	c1 = 5 c2 = 3 c3 = 3	c1 = 5 c2 = 7 c3 = 4	c1 = 4 c2 = 4 c3 = 3
Exist in 3 or more consecutive windows	c1 = 5 c2 = 6 c3 = 3	c1 = 5 c2 = 7 c3 = 4	c1 = 5 c2 = 6 c3 = 3

Table X. Results for a Single NN 3 Classes Set 3.

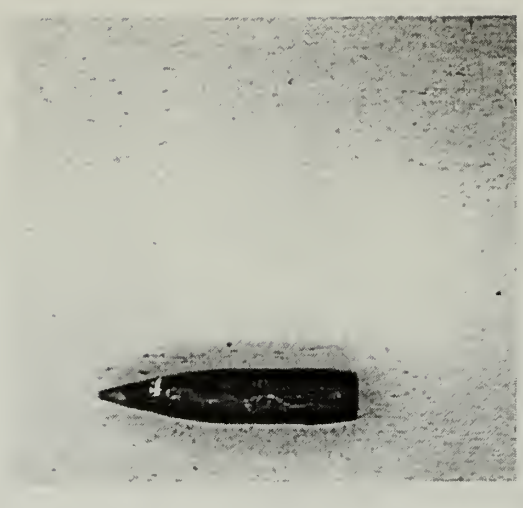
Set4 (3 Pictures, 2 Objects in each)	Correctly Identified	Correctly Located	Correctly Identified and Located
Min Error	c1 = 1 c2 = 2 c3 = 1	c1 = 1 c2 = 2 c3 = 2	c1 = 1 c2 = 2 c3 = 1

Table XI. Results for a Single NN 3 Classes Set 4.

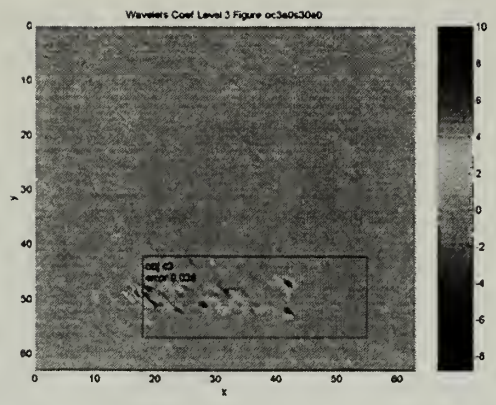
correctly located and 173 (71.2%) are classified correctly for location, angle in addition to identification.

- Out of the 63 samples in Test Set #3, 31 (49.2%) are classified correctly by object identification, 35 (55.6%) with correct angle information, 40 (63.5%) correctly located and 21 (33.3%) are classified correctly for location, angle in addition to identification.

- Out of the 18 samples in Test Set #4, 12 (66.7%) are classified correctly by object identification, 12 (66.7%) with correct angle information, 13(72.2%) correctly located and 8 (44.4%) are classified correctly for location, angle in addition to identification.



(a) Object Class 3



(b) Wavelet Domain for (a)

Figure 22. Location Results for Set #3 Three-Class Case

Set2 (27 Pictures) (Angle = 0°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 27 c2 = 22 c3 = 24	c1 = 27 c2 = 27 c3 = 25	c1 = 27 c2 = 22 c3 = 23	c1 = 27 c2 = 24 c3 = 25	c1 = 27 c2 = 22 c3 = 23
Formula	c1 = 27 c2 = 22 c3 = 24	c1 = 27 c2 = 27 c3 = 25	c1 = 27 c2 = 22 c3 = 23	c1 = 27 c2 = 24 c3 = 27	c1 = 27 c2 = 22 c3 = 23
Exist in 3 or more consecutive windows	c1 = 27 c2 = 25 c3 = 25	c1 = 27 c2 = 27 c3 = 26	c1 = 27 c2 = 24 c3 = 25	c1 = 27 c2 = 27 c3 = 25	c1 = 27 c2 = 24 c3 = 25

Table XII. Results for a Single NN 9 Classes Set 2 Angle 0°

Set2 (27 Pictures) (Angle = 45°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 8 c2 = 17 c3 = 17	c1 = 18 c2 = 18 c3 = 14	c1 = 8 c2 = 12 c3 = 13	c1 = 27 c2 = 23 c3 = 22	c1 = 8 c2 = 12 c3 = 13
Formula	c1 = 8 c2 = 17 c3 = 17	c1 = 18 c2 = 18 c3 = 14	c1 = 8 c2 = 12 c3 = 13	c1 = 27 c2 = 23 c3 = 22	c1 = 8 c2 = 12 c3 = 13
Exist in 3 or more consecutive windows	c1 = 12 c2 = 21 c3 = 17	c1 = 22 c2 = 22 c3 = 17	c1 = 12 c2 = 17 c3 = 15	c1 = 27 c2 = 25 c3 = 22	c1 = 12 c2 = 17 c3 = 15

Table XIII. Results for a Single NN 9 Classes Set 2 Angle 45°

Set2 (27 Pictures) (Angle = 90°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 26 c2 = 21 c3 = 22	c1 = 27 c2 = 25 c3 = 24	c1 = 26 c2 = 21 c3 = 21	c1 = 27 c2 = 26 c3 = 24	c1 = 26 c2 = 21 c3 = 21
Formula	c1 = 26 c2 = 21 c3 = 22	c1 = 27 c2 = 25 c3 = 24	c1 = 26 c2 = 21 c3 = 21	c1 = 27 c2 = 26 c3 = 24	c1 = 26 c2 = 21 c3 = 21
Exist in 3 or more consecutive windows	c1 = 26 c2 = 24 c3 = 24	c1 = 27 c2 = 26 c3 = 24	c1 = 26 c2 = 24 c3 = 23	c1 = 27 c2 = 27 c3 = 25	c1 = 26 c2 = 24 c3 = 23

Table XIV. Results for a Single NN 9 Classes Set 2 Angle 90°

Set3 (7 Pictures) (Angle = 0°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 1 c2 = 4 c3 = 4	c1 = 4 c2 = 6 c3 = 6	c1 = 1 c2 = 3 c3 = 4	c1 = 6 c2 = 6 c3 = 2	c1 = 1 c2 = 3 c3 = 2
Formula	c1 = 1 c2 = 4 c3 = 4	c1 = 4 c2 = 6 c3 = 6	c1 = 1 c2 = 3 c3 = 4	c1 = 7 c2 = 7 c3 = 2	c1 = 1 c2 = 3 c3 = 2
Exist in 3 or more consecutive windows	c1 = 4 c2 = 5 c3 = 5	c1 = 7 c2 = 7 c3 = 6	c1 = 4 c2 = 4 c3 = 5	c1 = 7 c2 = 7 c3 = 4	c1 = 4 c2 = 4 c3 = 3

Table XV. Results for a Single NN 9 Classes Set 3 Angle 0°

Set3 (7 Pictures) (Angle = 45°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 1 c2 = 3 c3 = 3	c1 = 1 c2 = 3 c3 = 2	c1 = 0 c2 = 3 c3 = 2	c1 = 4 c2 = 4 c3 = 4	c1 = 0 c2 = 3 c3 = 2
Formula	c1 = 1 c2 = 3 c3 = 3	c1 = 1 c2 = 3 c3 = 2	c1 = 0 c2 = 3 c3 = 2	c1 = 4 c2 = 4 c3 = 4	c1 = 0 c2 = 3 c3 = 2
Exist in 3 or more consecutive windows	c1 = 2 c2 = 4 c3 = 3	c1 = 2 c2 = 4 c3 = 3	c1 = 1 c2 = 4 c3 = 2	c1 = 5 c2 = 4 c3 = 5	c1 = 1 c2 = 4 c3 = 2

Table XVI. Results for a Single NN 9 Classes Set 3 Angle 45°

Set3 (7 Pictures) (Angle = 90°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 4 c2 = 6 c3 = 5	c1 = 3 c2 = 6 c3 = 4	c1 = 3 c2 = 6 c3 = 3	c1 = 4 c2 = 7 c3 = 3	c1 = 3 c2 = 6 c3 = 1
Formula	c1 = 4 c2 = 6 c3 = 5	c1 = 3 c2 = 6 c3 = 4	c1 = 3 c2 = 6 c3 = 3	c1 = 4 c2 = 7 c3 = 3	c1 = 3 c2 = 6 c3 = 1
Exist in 3 or more consecutive windows	c1 = 7 c2 = 6 c3 = 6	c1 = 5 c2 = 7 c3 = 6	c1 = 5 c2 = 6 c3 = 5	c1 = 6 c2 = 7 c3 = 7	c1 = 5 c2 = 6 c3 = 4

Table XVII. Results for a Single NN 9 Classes Set 3 Angle 90°

Set4 (3 Pictures, 2 Objects in each) (Angle = 0°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 2 c2 = 1 c3 = 2	c1 = 2 c2 = 1 c3 = 1	c1 = 2 c2 = 1 c3 = 1	c1 = 2 c2 = 1 c3 = 1	c1 = 2 c2 = 1 c3 = 1

Table XVIII. Results for a Single NN 9 Classes Set 4 Angle 0°

Set4 (3 Pictures, 2 Objects in each) (Angle = 45°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 0 c2 = 1 c3 = 2	c1 = 1 c2 = 1 c3 = 1	c1 = 0 c2 = 1 c3 = 1	c1 = 1 c2 = 2 c3 = 1	c1 = 0 c2 = 1 c3 = 0

Table XIX. Results for a Single NN 9 Classes Set 4 Angle 45°

Set4 (3 Pictures, 2 Objects in each) (Angle = 90°)	Correct Class Id	Correct Angle	Correct Class Id and Angle	Correctly Located	Correctly Identified and Located
Min Error	c1 = 1 c2 = 1 c3 = 2	c1 = 2 c2 = 2 c3 = 1	c1 = 1 c2 = 1 c3 = 1	c1 = 2 c2 = 2 c3 = 1	c1 = 1 c2 = 1 c3 = 1

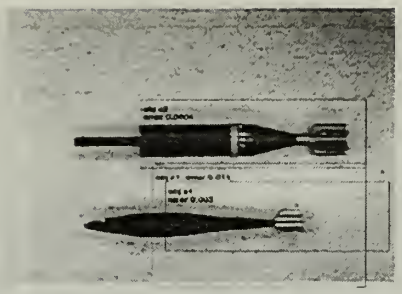
Table XX. Results for a Single NN 9 Classes Set 4 Angle 90°

In Figure 23 samples from set # 4 are shown. Figure 23 (a) shows objects c1 and c2 at angle 0°, the boxes and text superimposed indicate the location and identity information produced by the NN. All existing answers for cases A3 through A8 (See Section 2.b of this chapter) are displayed. Figure 23 (b) is the third decomposition level of the wavelet domain, again with the answers boxes and text superimposed. Figure 23 (c), (d) and (e), (f) are pairs corresponding to other samples of set #4.

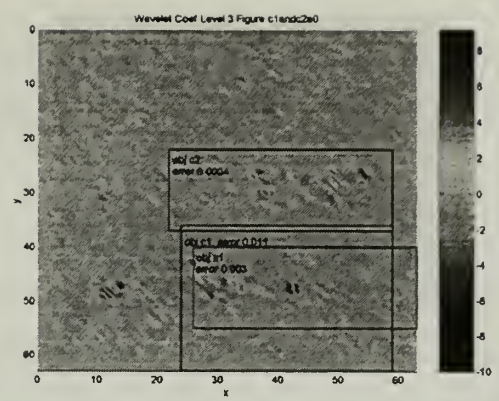
In Figure 24 shows UXOs not in our training database. These images were used to test the occurrence of false positives by our system. Note that false positives are produced by our system. However, it is interesting to note that our system does do a good job at localizing these objects in the scene.

3. Timing

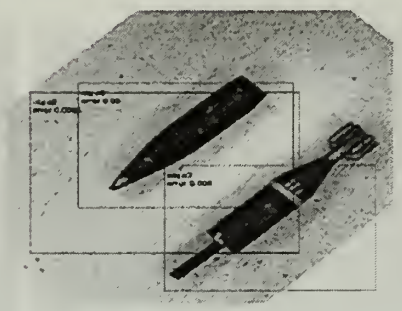
One minute is the typical time it takes to process a single scene image (scanning plus NN processing). This speed could be greatly improved by implementing the recognition system in a language like C rather than Lisp which is an interpretive language and requires a rather long loading and running phases (approximately 3 times slower than C). Another improvement could be made in the scheme by integrating the template scanning scheme currently implemented in C with the Neural Network System and stopping template scanning when the first acceptable template is found.



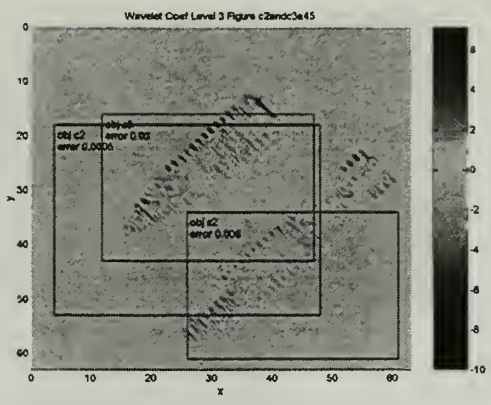
(a) Objects Class 1 and 2 Angle 0°



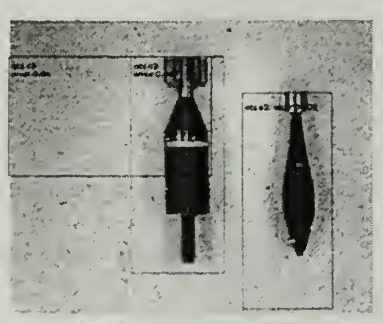
(b) Wavelet Domain for (a)



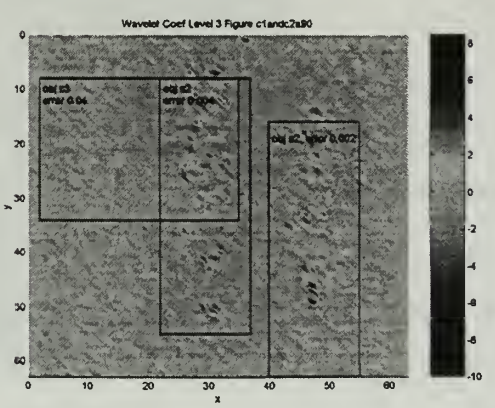
(c) Objects Class 2 and 3 Angle 45°



(d) Wavelet Domain for (c)

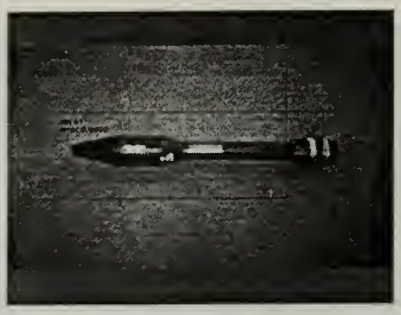


(e) Objects Class 1 and 2 Angle 90°

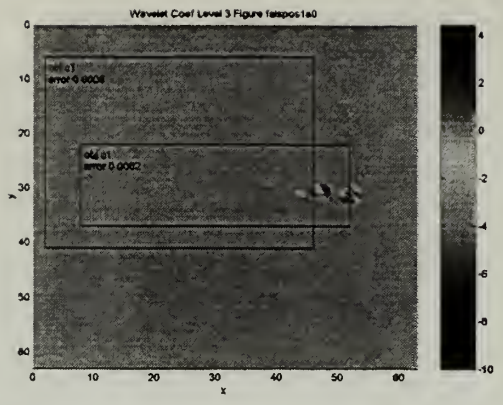


(f) Wavelet Domain for (e)

Figure 23. Location Results for Set #4 Nine-Class Case

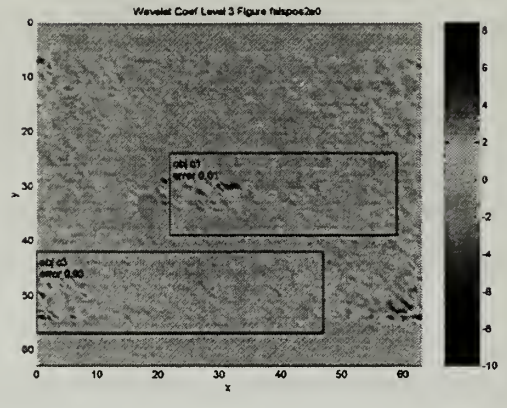


(a) Sample 1 Identified as Class1



(b) Wavelet Domain for (a)

(c) Sample 2 Identified as Class 1



(d) Wavelet Domain for (c)

Figure 24. Samples of False Positive Results

4. Discussion

This work shows that it is possible to use computer vision techniques for UXO detection with good accuracy for a simple scene and limited accuracy for complicated scenes. The results indicate the scheme is much better at localizing than identifying objects. The use of higher level, more global features for identification may be necessary, instead of the local ones used in this research.

V. AUTONOMOUS VEHICLE NAVIGATION ISSUES

A. INTRODUCTION

This chapter will give an overview of autonomous vehicle motion and positioning. This is a different but essential component of creating a mobile robot capable of UXO detection in the field. All motion planning theory presented is based on the work in references 4,5,6. "Yamabico" (Figure 25) is the robot developed by Professor Kanayama, which is used to test some of the concepts presented here.

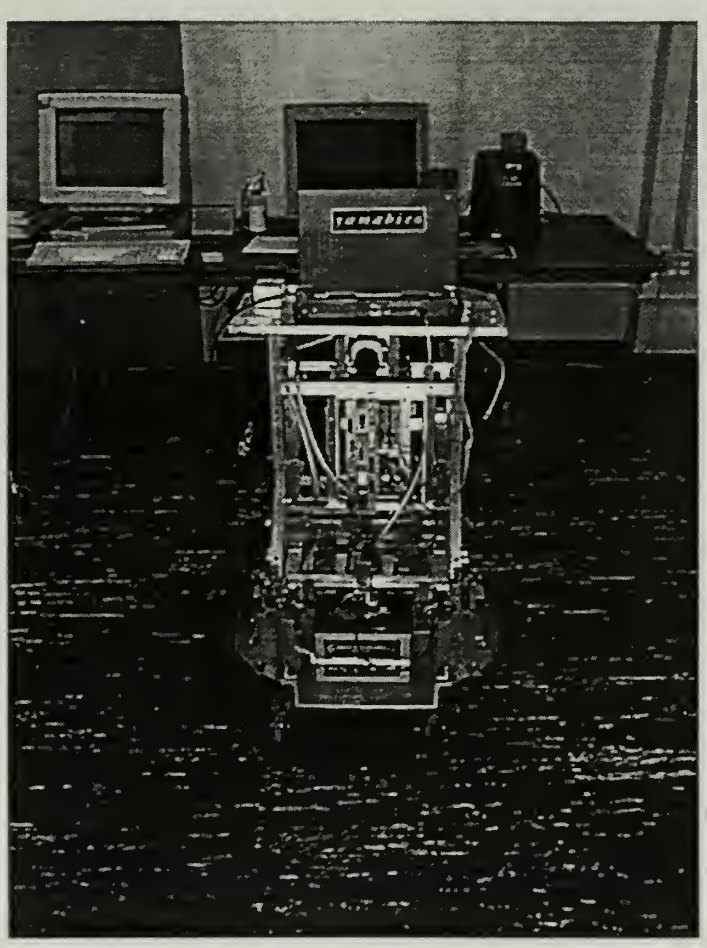


Figure 25. Robot "Yamabico"

This work is in support of the navigation system of "Yamabico". Currently,

it moves in an indoor environment. This thesis made the following contributions to the robot's system.

1. Development of a more general line fitting algorithm used in the edge detection system.
2. Creation of a 3D feature map of an indoor environment.
3. Development of an algorithm to extract the visible features (vertical lines) of a 2D map, given a viewpoint.
4. Implementation of the steering theory [Ref. 4] in "Yamabico's" motion system.

B. POSITIONING

There are several ways of positioning a vehicle in the real world. If one is not using an outside navigation system such as GPS, solving this problem for an outdoor vehicle can be very complex. Our research will deal with indoor navigation for an autonomous vehicle. It may be possible to extend techniques used for indoor navigation to outdoor navigation. Two stages are necessary in performing positioning in indoor environment. The first involves using computer vision techniques to find landmarks in the scene and the second involves matching these to landmarks in a 3D map of the robot's environment. The work done in this thesis touches upon each of these stages. In Section 1, below, we discuss a method that can be used to fit straight lines to edge data. Such lines are commonly used as landmarks.

A second part of this work described in section 2, presents a methodology to constructs a 3D model using linear landmarks.

1. Weighted Line Fitting Algorithm

One way, to help a robot to position itself autonomously in a indoor environment, is by processing visual information. The robot can process a picture, find edges representing landmarks, match them with a model of the world, and use the resulting positioning information to correct its inertial navigation. Edge detection

can be performed by several different operators. We have chosen the Sobel operator [Ref. 7].

After the edges have been extracted from a picture, we need to fit the edge pixels to a straight line. This is accomplished by a form of Least Squares Fitting.

Based on Chapter 10 of [Ref. 21], we show below the equations for finding a straight line given a set of points on it. The weight factor w present in the equations, is the magnitude of each pixel calculated by the Sobel operator signifying the strength of the edge. The program previously written to calculate edges of a picture, was modified to support the inclusion of this factor. So in our case, w will be

$$w = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{V.1}$$

where f is a function that denotes the gradient of the greyscale value of a pixel located at (x_i, y_i). This is the result of the application of the Sobel operator over that pixel.

To detect a straight line segment that fits a set of edge points, we want the line which minimizes the distance of the edge to the line. We represent this distance as the sum of squared distances between all points and the line in question. The best line is found by continuously modifying the description of the line segment to best fit the data using a least squares fitting algorithm.

We will derive now a series of Lemmas ending in a proposition that is the heart of our new algorithm to perform a weighted line fitting which we use to detect our straight line segments. Those segments will, in the future, be used in our mobile robot positioning system. Recall, we take the weights from the edge values. Points which greatly increase the error are assumed to not belong to the line under consideration. Let

$$R = \{p_1, \cdots, p_n\} = \{(x_1, y_1), \cdots, (x_n, y_n)\} \quad n \geq 2 \tag{V.2}$$

be a set of n points that are not all equal. The moments m_{jk} of R are defined as follows.

$$m_{00} = \sum_{i=1}^{n} w_i \tag{V.3}$$

$$m_{10} = \sum_{i=1}^{n} w_i x_i \tag{V.4}$$

$$m_{01} = \sum_{i=1}^{n} w_i y_i \tag{V.5}$$

$$m_{20} = \sum_{i=1}^{n} w_i x_i^2 \tag{V.6}$$

$$m_{11} = \sum_{i=1}^{n} w_i x_i y_i \tag{V.7}$$

$$m_{02} = \sum_{i=1}^{n} w_i y_i^2 \tag{V.8}$$

The centroid $C = (\mu_x, \mu_y)$ of the set R is given by

$$C = (\mu_x, \mu_y) = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right). \tag{V.9}$$

The secondary moments about the centroid are defined as

$$M_{20} = \sum_{i=1}^{n} w_i (x_i - \mu_x)^2 \tag{V.10}$$

$$M_{11} = \sum_{i=1}^{n} w_i (x_i - \mu_x)(y_i - \mu_y) \tag{V.11}$$

$$M_{02} = \sum_{i=1}^{n} w_i (y_i - \mu_y)^2 \tag{V.12}$$

The following relations are easily verified.

Lemma 1

$$\begin{aligned} M_{20} &= \sum_{i=1}^{n} w_i (x_i - \mu_x)^2 \\ &= \sum_{i=1}^{n} (w_i x_i^2 - 2 w_i x_i \mu_x + w_i \mu_x^2) \\ &= m_{20} - 2 \frac{m_{10}}{m_{00}} m_{10} + \left(\frac{m_{10}}{m_{00}} \right)^2 m_{00} \\ &= m_{20} - 2 \frac{m_{10}^2}{m_{00}} + \frac{m_{10}^2}{m_{00}} \\ M_{20} &= m_{20} - \frac{{m_{10}}^2}{m_{00}} \\ M_{11} &= \sum_{i=1}^{n} w_i (x_i - \mu_x)(y_i - \mu_y) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n}(w_i x_i y_i - w_i x_i \mu_y - w_i y_i \mu_x + w_i \mu_x \mu_y) \\
&= m_{11} - m_{10}\mu_y - m_{01}\mu_x + m_{00}\mu_x \mu_y \\
&= m_{11} - \frac{m_{01}}{m_{00}} m_{10} - \frac{m_{10}}{m_{00}} m_{01} + \frac{m_{10} m_{01}}{m_{00}^2} m_{00} \\
M_{11} &= m_{11} - \frac{m_{10} m_{01}}{m_{00}} \\
M_{02} &= \sum_{i=1}^{n} w_i (y_i - \mu_y)^2 \\
&= \sum_{i=1}^{n}(w_i y_i^2 - 2 w_i y_i \mu_y + w_i \mu_y^2) \\
&= m_{02} - 2\frac{m_{01}}{m_{00}} m_{01} + \left(\frac{m_{01}}{m_{00}}\right)^2 m_{00} \\
&= m_{02} - 2\frac{m_{01}^2}{m_{00}} + \frac{m_{01}^2}{m_{00}} \\
M_{02} &= m_{02} - \frac{{m_{01}}^2}{m_{00}}
\end{aligned}
$$

Let us consider a ray starting from the origin $O = (0, 0)$ with a direction of α, where $\alpha \in [-\pi/2, \pi/2]$. For an arbitrary point $p = (x, y)$, let H be the closest point on the ray from p.

Lemma 2 *When a ray of a direction α and a point p is given, the distance ρ from O to H is given by*

$$\rho = x \cos\alpha + y \sin\alpha. \tag{V.13}$$

The "distance" ρ may be negative or zero.

Using this result, we adopt the parametric representation of a line by its normal direction α and the distance r from the origin O (α and r are constants). Actually, this line

$$L = (r, \alpha) \tag{V.14}$$

is the set of points (x, y) which satisfies the relation

$$x \cos\alpha + y \sin\alpha = r. \tag{V.15}$$

This representation has a striking advantage as opposed to the normal method of using a formula $y = f(x)$, as this parametric method has no singularity with respect to vertical lines.

Lemma 3 *The signed distance (or residual) δ_i from point $p_i = (x_i,\ y_i)$ to the line $L = (r, \alpha)$ is*

$$\delta_i = x_i \cos\alpha + y_i \sin\alpha - r. \tag{V.16}$$

Proposition 1 *For a set of points*

$$R = \{p_1, \cdots, p_n\} = \{(x_1, y_1), \cdots, (x_n, y_n)\}, \tag{V.17}$$

a line $L = (r, \alpha)$ with

$$\alpha = \frac{1}{2} atan2(-2M_{11},\ M_{02} - M_{20}) \tag{V.18}$$

$$r = \frac{m_{10}}{m_{00}} \cos\alpha + \frac{m_{01}}{m_{00}} \sin\alpha = \mu_x \cos\alpha + \mu_y \sin\alpha \tag{V.19}$$

makes the sum of residuals δ_i minimum.

Proof.

The sum of the squares of all the residuals is

$$S = \sum_{i=1}^{n} w_i \Big((x_i \cos\alpha + y_i \sin\alpha) - r \Big)^2 \tag{V.20}$$

Since the line which best fits the set of points is supposed to minimize S, the optimum line (r, α) must satisfy

$$\frac{\partial S}{\partial \alpha} = \frac{\partial S}{\partial r} = 0 \tag{V.21}$$

Thus,

$$\begin{aligned} \frac{\partial S}{\partial r} &= -2 \sum_{i=1}^{n} w_i \Big((x_i \cos\alpha + y_i \sin\alpha) - r \Big) \\ &= 2 \left(r \left(\sum_{i=1}^{n} w_i \right) - \left(\sum_{i=1}^{n} w_i x_i \right) \cos\alpha - \left(\sum_{i=1}^{n} w_i y_i \right) \sin\alpha \right) \\ &= 2(r\ m_{00} - m_{10} \cos\alpha - m_{01} \sin\alpha) = 0 \end{aligned} \tag{V.22}$$

and

$$r = \frac{m_{10}}{m_{00}} \cos\alpha + \frac{m_{01}}{m_{00}} \sin\alpha = \mu_x \cos\alpha + \mu_y \sin\alpha \tag{V.23}$$

where r may be negative. Substituting r in Equation V.15 by Equation V.23, we obtain

$$\begin{aligned}
\frac{\partial S}{\partial \alpha} &= 2\sum_{i=1}^{n} w_i\Big((x_i-\mu_x)\cos\alpha+(y_i-\mu_y)\sin\alpha\Big)\Big(-(x_i-\mu_x)\sin\alpha+(y_i-\mu_y)\cos\alpha\Big) \\
&= 2\sum_{i=1}^{n} w_i\Big((y_i-\mu_y)^2-(x_i-\mu_x)^2\Big)\sin\alpha\cos\alpha \\
&\quad +2\sum_{i=1}^{n} w_i(x_i-\mu_x)(y_i-\mu_y)(\cos^2\alpha-\sin^2\alpha) \\
&= (M_{02}-M_{20})\sin 2\alpha+2M_{11}\cos 2\alpha=0 \qquad \text{(V.24)}
\end{aligned}$$

Therefore,

$$2\alpha = \text{atan2}(-2M_{11},\ M_{02}-M_{20}) \qquad \text{(V.25)}$$

□

Thus, the value of 2α is in the fourth quadrant, $[-\pi,\pi]$, and then $\alpha \in [-\pi/2,\pi/2]$.

a. Implementation

The program code developed is given in Appendix F. Figure 26 thru Figure 29 show some of the results obtained with and without the inclusion of weights. The threshold indicated is the value used after applying the Sobel operator, to decide if the pixel belongs to a landmark or not. The maximum ϕ difference shown, defines the maximum directional difference (in degrees) allowed between adjacent pixels, to be considered on the same line.

b. Results

Unfortunately, there is not a significant visual improvement when including weights to justify the added computational burden. However, we can not discard the possibility that for some specific cases that were not tested, this approach could produce better results. More experimentation is needed.

Figure 26. Results without Weights, Threshold = 100, Max ϕ Difference = 22

Figure 27. Results with Weights, Threshold = 100, Max ϕ Difference = 22

Figure 28. Results without Weights, Threshold = 75, max ϕ difference = 22

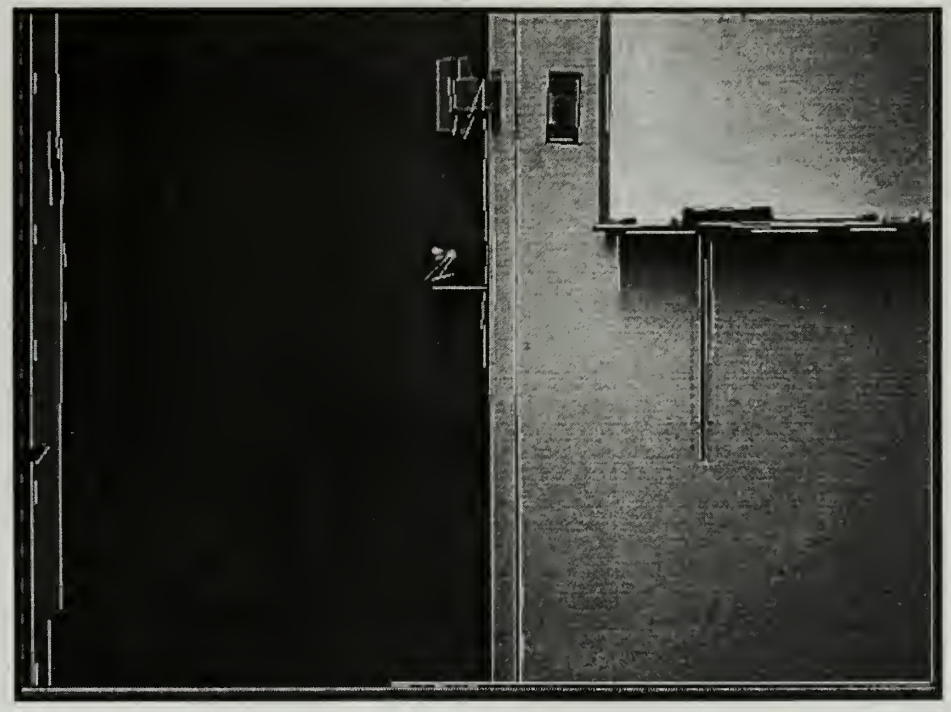


Figure 29. Results with Weights, Threshold = 75, max ϕ difference = 22

2. 3D MODELING

Another contribution made to the "Yamabico" mobile robot system, is the modification of a 3D modeling program to simulate rendering of views by a camera that can be positioned at different locations in an indoor 3D model. The 3D modeling program is implemented in Lisp and is listed in Appendix G. Figure 30 and Figure 31, depict some frozen frames, rendered camera views. The 3D model consists of connected edges.

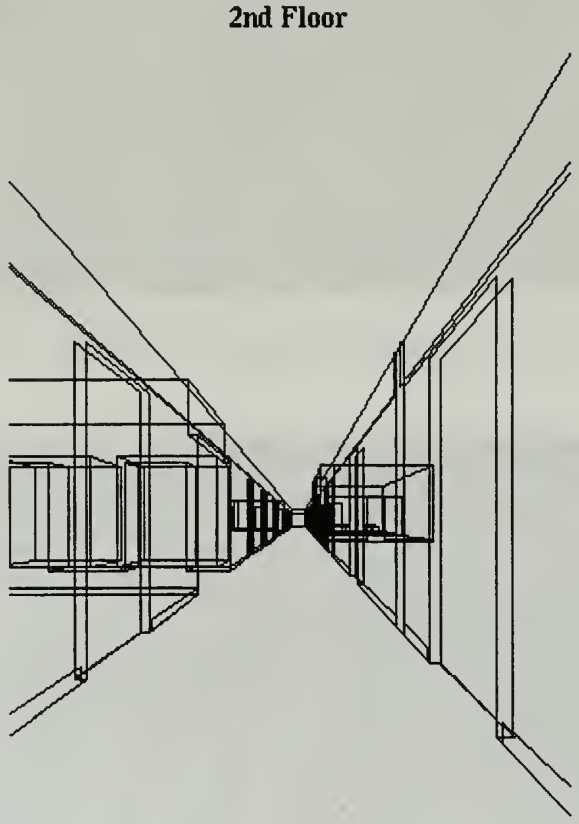


Figure 30. Frozen Frame 1

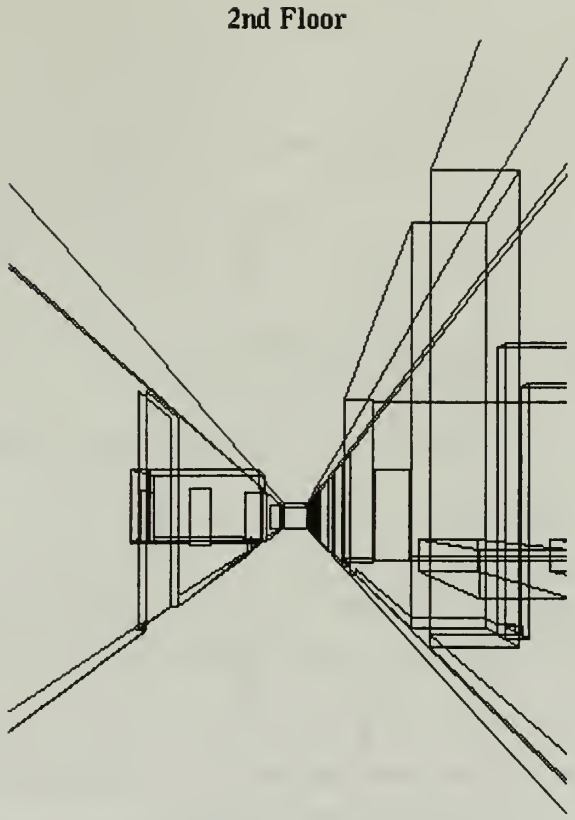


Figure 31. Frozen Frame 2

a. Extraction of 2D Features from Rendered Model Views

Once a model of the world is constructed, we still need to find what lines in the model are visible, at different viewpoints. These features can be matched to features extracted from a real scene to update the robot's position. Recall that we detect straight line segments. After analyzing the problem, we decided that just extracting features from a 2D top view of the 3D map is sufficient. This will yield the visible vertical lines of the world, which can be matched with the scene's vertical edges. Appendix H contains the program to find the visible vertical landmarks of a given model. Although still not perfect, the program produced very good results for a large set of cases, as shown in Figure 32 and Figure 33. The C represents the camera position and the diamonds ($\diamond$) represent the visible vertical lines.

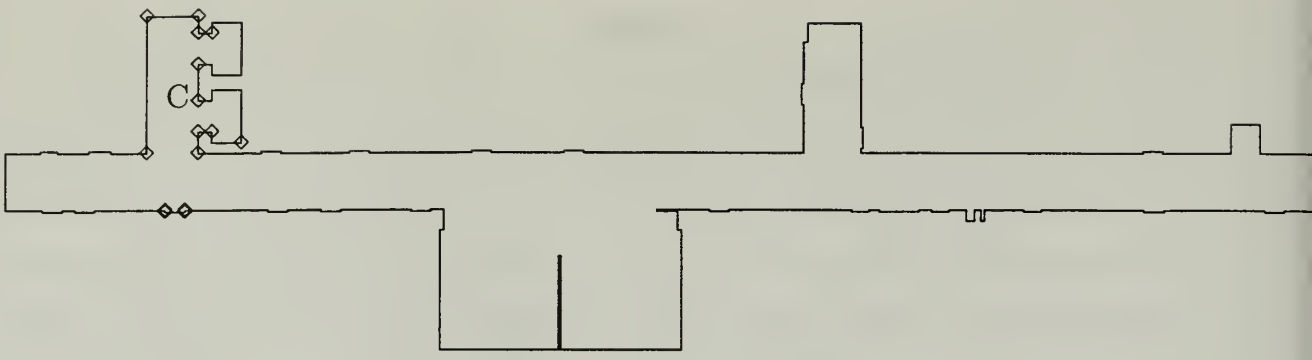


Figure 32. Second Floor

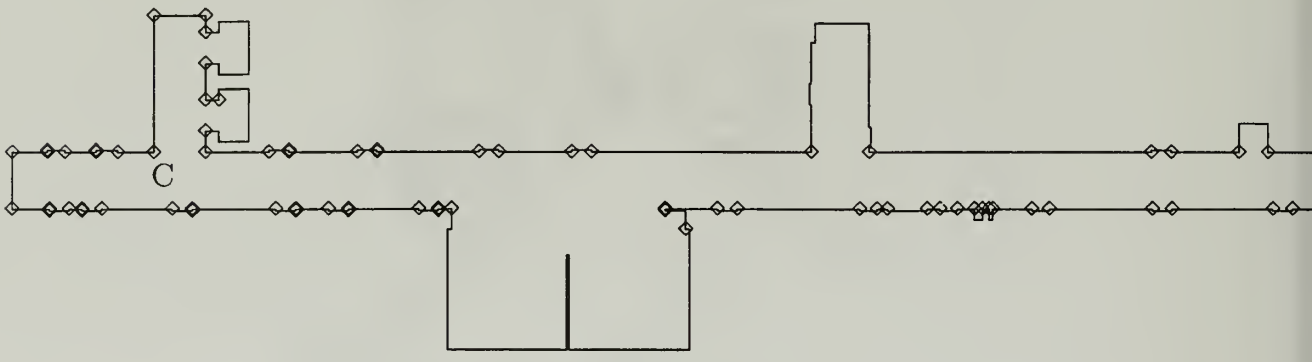


Figure 33. Second Floor

C. MOTION PLANNING

The non-linear control theory presented here is based on Professor's Kanayama tracking methods presented in [Ref. 4, 5, 6]. All the simulation results presented were actually tested on the autonomous vehicle "Yamabico" (Figure 25).

1. Linear and Circular Tracking

Suppose we want a vehicle in a certain position to move towards a line, as shown in Figure 34.

Let's now define a vehicle's state as a *configuration* [Ref. 4],

$$q = (p, \theta, \kappa), \tag{V.26}$$

where p denotes its (x, y) coordinates, θ is its orientation and κ is its instantaneous path curvature.

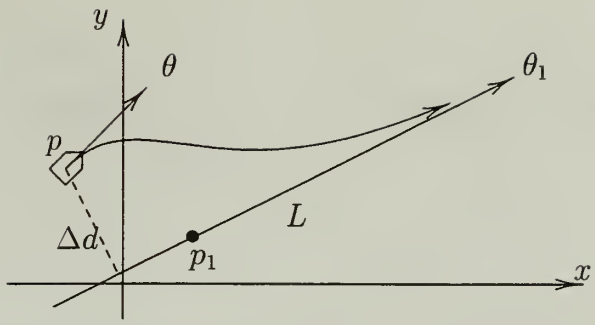


Figure 34. Principle of Path Tracking

The equation that describes motion is given by

$$\frac{d\kappa}{ds} = -a\kappa - b(\theta - \theta_1) - c\Delta d, \tag{V.27}$$

where s is the arc length, a, b, c are positive constants, and Δd is the "signed" distance from p to L. This equation with its negative terms represents a negative feedback rule that we called the *steering function.* The first negative term, $-a\kappa$ is a feedback term (a damping factor) for the curvature, the second term $-b(\theta - \theta_1)$ is a feedback term for the angle error, and the third term $-c\Delta d$ is a feedback term for the *positional error.*

As shown in [Ref. 4], a, b, c can be defined as

$$a = 3k, \quad b = 3k^2, \quad c = k^3, \tag{V.28}$$

where k the *gain* of the steering function. A better way of visualizing the effects of k in the steering function, is defined by

$$\sigma = \frac{1}{k}, \tag{V.29}$$

where σ is called *smoothness.* In other words, if σ is large we have a smooth trajectory and if σ is small we have a sharper trajectory, as shown in Figure 35.

For a circular tracking situation as presented in Figure 36, the steering has to be slightly modified (Equation V.30).

$$\frac{d\kappa}{ds} = -a(\kappa - \kappa_1) - b(\theta - \theta_1) - c\Delta d. \tag{V.30}$$

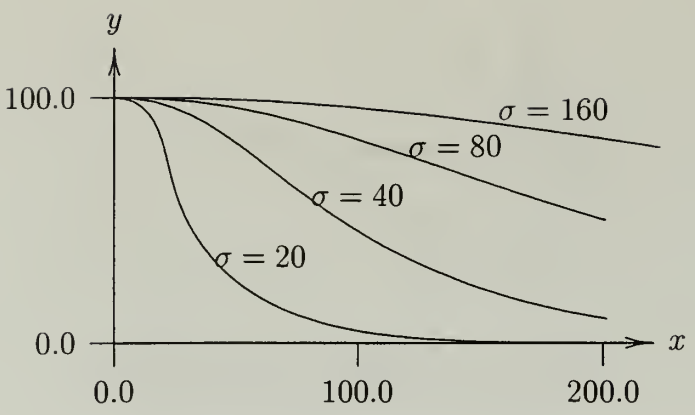


Figure 35. Effect of Smoothness

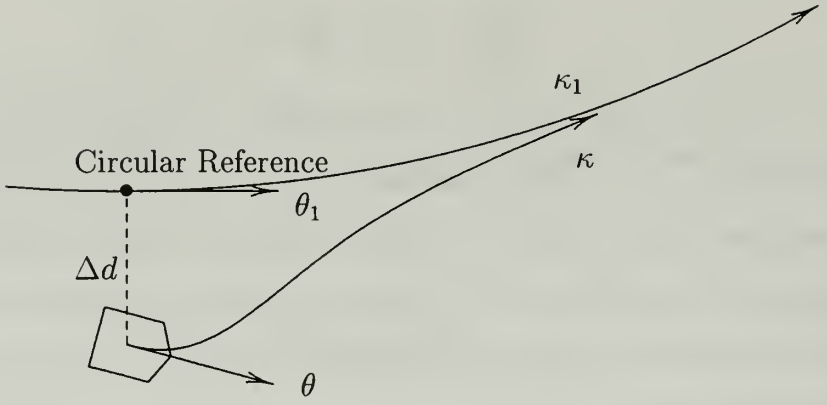


Figure 36. Tracking a Circle

The curvature κ_1 is now the curvature of the circle we are tracking and is equal to $\frac{1}{R}$, where R is the radius.

As proven in [Ref. 5] we have

$$a = 3k, \tag{V.31}$$

$$b = 3k^2 - {\kappa_1}^2, \tag{V.32}$$

$$c = k^3 - 3k{\kappa_1}^2, \tag{V.33}$$

which also holds for the linear tracking. The parameter σ is still defined in the same way as before and has similar effect as shown in Figure 37.

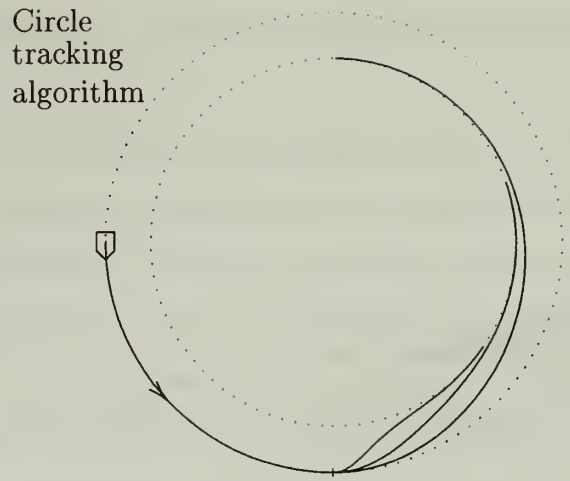


Figure 37. Circle Tracking for $\sigma = 1, 2, 4$ (from Left to Right)

2. Neutral Switching

Although the motion resulting from the use of the steering function is very smooth, finding the right point to leave a path to transition to another is a problem.

[Ref. 6] states that for a smoother path change, the leaving point must be the one where the steering function changes its sign. In other words,

$$\frac{d\kappa}{ds} = -a\Delta\kappa - b\Delta\theta - c\Delta d = 0. \tag{V.34}$$

For a situation like Figure 38 where $\Delta\kappa = 0$ and $\Delta\theta = -\frac{\pi}{2}$ we have

$$\Delta d = -\frac{b\Delta\theta}{c} = 3\Delta\theta\sigma = 4.712\sigma. \tag{V.35}$$

We have for Figure 38 the following leaving points for the same smoothness ($\sigma = 1.0$): $(0, 7)$, $(0, 4.712)$, $(0, 3.5)$, $(0, 3)$, $(0, 2.5)$, $(0, 2)$, $(0, 1.5)$ and $(0, 1)$.

In Figure 39, we show the curvature plots corresponding to Figure 38. We just labeled the curves for the end leaving points (p_1 and p_2) and for the neutral switching point (p_n), to preserve readability in the plots. One can easily see that the curvature for the neutral switching point (curve p_n) has smaller values and also that its sign never changes (compare with curves p_1 and p_2). This is the reason why the neutral switching principal provides smoother control of a vehicle.

Figure 38. Trajectory for Leaving Points at $y = 7$, 4.712, 3.5, 3, 2.5, 2, 1.5 and 1.

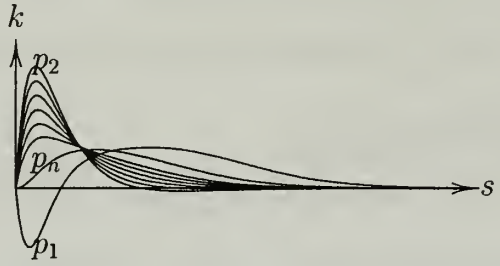


Figure 39. Curvature Plots for Leaving Points at $y = 7$, 4.712, 3.5, 3, 2.5, 2, 1.5 and 1.

Notice that for the leaving points $y = 4.712$ (neutral switching) and $y = 3.5$, the curves in the $x \times y$ plot are very close ($y = 4.712$ is the furtherest curve from the origin and $y = 3.5$ is the one that follows it). However, in Figure 39 ($y = 3.5$ is the one above p_n), the plot $s \times k$, or $path - length \times curvature$, shows a big difference.

An interesting point found during this research is that for the leaving points after neutral switching, their paths have a maxima curvature close to the same point. Another remarkable issue is that those curves also cross together at the same path length value, meaning that no matter how late you leave, you will always have the same instant curvature at the same traveled distance (for a fixed σ). These two characteristics of the neutral switching control theory were discovered at the end of this research. They are characteristics that deserve future examination. Appendix I shows the high level C code used to control "Yamabico's" motion, using this theory.

VI. CONCLUSION AND FUTURE WORK

A. RECOGNITION/LOCALIZATION SYSTEM

This work shows that it is possible to use computer vision techniques for UXO detection with good accuracy for simple scenes and limited accuracy for complicated scenes. Both the performance and the accuracy of the system can be improved in a number of ways. The greatest need is for a much larger set of training data which will improve the accuracy of the system. Secondly, experiments in the extraction of high-order features such as boundaries could lead to improved recognition results at the cost of increased processing time. The results indicate that our scheme is much better at locating than identifying objects. The use of more global features for identification may be necessary, instead of the local ones used in this research.

A significant increase in performance could be achieved through the fusion of multiple sensor data such as that from magnetometers and video. It is important to note that the techniques described here including the wavelet feature extraction and the neural network Stages could be used with other kinds of data.

A better refined template scanning scheme is another avenue of future research. Templates could be created to optimally detect the objects in the database. Since our windows are all rectangular (oriented along the axis), more errors occurred for objects at 45°. A 45° oriented template would greatly improve the results for such situation.

In conclusion, this work has shown that the proposed recognition system utilizing Wavelet Feature Extraction and Neural Networks is promising for UXO detection. Also, this work has demonstrated that images can be used as sensory input for UXO detection. However, more experimental work is needed to further refine the system.

B. NAVIGATION SYSTEM

The work done in Chapter V of this thesis can be used to improve a robot's navigation. This is especially true with regards to the neutral switching theory, which was implemented for the first time on a robot ("Yamabico") and resulted in very smooth motion.

Basically we found the following important observations and facts in the area of autonomous navigation:

- The inclusion of gradient values (weights) in the edge detection algorithm for grayscale digital images does not significantly improve its results. Actually, this result is attractive to us, because the computationally inexpensive method we are current using is enough to obtain good results.
- The visibility algorithm gave satisfactory results as expected. The work that should immediately follows is the integration of the image understanding capability with this visibility capability into the current "Yamabico" software system.
- The success of the neutral switching function is a big mile stone for autonomous vehicle research. Complex motion behavior are made possible for "Yamabico" through this algorithm. As a future plan, more detailed functions should be designed and implemented.

APPENDIX A. LISP CODE FOR TRAINING A NEURAL NETWORK (NN) USING BACK-PROPAGATION

```
;*****************************************************************
; File: neuron.lisp
; Name:Jader Gomes da Silva Filho
; Date:01/97
; Operating Environment: SUN
; Language: LISP
;*****************************************************************

;;-------------------------------------------------
;; Initial values for the weights and global variable
;;

(defun create_weights (inp out)

  ;;constant for random starting weight
  ;;
  (setf RSW 1.0)

  ;; initial value for beta
  (setf beta 0.1)

  (setf SAVED_ERROR  (list_of (length inp) (list_of (length (car out)) 0.5)))

  (setf inp_wlist '())
  (do (( i 1 (+ i 1)))
      (( > i (length (car inp))) 'DONE)
      (setf inp_wlist(append inp_wlist
     (list
      (make_random_weight_list
       (length (car inp)) RSW)))))
  (setf hid_wlist '())
  (do (( i 1 (+ i 1)))
      (( > i (length (car out))) 'DONE)
      (setf hid_wlist(append hid_wlist
     (list
      (make_random_weight_list
       (length (car inp)) RSW)))))
  )

;;-------------------------------------------------
;;Genetates a list of repeated inputs of the same size of
;;the weigth list.
```

```
;;
(defun make_input_list (input w_list)
  (list_of (length w_list) input))

;;-------------------------------------------------
;; this provides a standard activation
;; function for a neural net--the logistic
;; sigmoid function
;; f(x) = 1 / (1 + e (-x))
;;
;;
(defun sigmoid (x)
   (/ ( + 1 (exp (- x)))))

;;-------------------------------------------------
;; this does summation on a vector
;;
;; > (summation vector)
;; > (summation '(2.0 3.0 2.6))
;; 7.6
;;
(defun summation (vector)
   (eval
        (cons '+ vector)))

;;-------------------------------------------------
;; this does multiplication of two vectors
;;
;; > (vector_multiply '(0 1 1) '(-4.8 5.2 -2.3))
;; (0 5.2 -2.3)
;;
(defun vector_multiply (vector1 vector2)
     (mapcar #'* vector1 vector2))

;;-------------------------------------------------
;; this does multiplication of three vectors
;;
;;
(defun vector_multiply3 (vector1 vector2 vector3)
     (mapcar #'* vector1 vector2 vector3))

;;-------------------------------------------------
```

```
;; this makes a list n elements long of elt
;;
;; > (list-of 3 2.0)
;; (2.0 2.0 2.0)
;;
(defun list_of (n elt)
   (if (zerop n)
        nil
       (cons elt (list_of ( - n 1) elt))))

;;-------------------------------------------------
;; this makes a weight list n elements long
;; of random weights between -RSW and RSW (random starting weight)
;;
(defun make_random_weight_list (n rsw)
   (vector_sum (mapcar #'random (list_of n (* 2 rsw)))
       (list_of n (- rsw))))

;;-------------------------------------------------
;; this does sum of two vectors
;;
;; > (vector_sum '(0 1 1) '(-4.8 5.2 -2.3))
;; (-4.8 6.2 -1.3)
;;
(defun vector_sum (vector1 vector2)
     (mapcar #'+ vector1 vector2))

;;-------------------------------------------------
;;Evaluates one node according to a list of inputs
;;and a list of weigths for that node
;; > (node_eval '(0 0 1) '(-4.8  4.6 -2.6))
;; 0.06913843

(defun node_eval (inputs weigths)
  (sigmoid (summation (vector_multiply inputs weigths))))

;;-------------------------------------------------
;;Evaluates one level, generating a list of values
;;that will be the input for the next level
;;
;; > (level_eval '(0 0) inp_wlist)
;; (0.06913843 0.03916572)
```

```
(defun level_eval (input w_list)
  (mapcar #'node_eval (make_input_list input w_list) w_list))


;;-------------------------------------------------
;; CALCULATE ERROR VECTOR FOR OUTPUT NODES
;; this compares calculated output with
;; expected output. And save the errors in the
;; variable SAVED_ERROR_OUT
;;
;; > (calc_output_error '(1.0 0.0) '(0.88 0.13) )
;; (+0.12 -0.13)
;;
(defun calc_output_error (exp_output calc_output)
  (setf SAVED_ERROR_OUT
(mapcar #'- exp_output calc_output)))

;;-------------------------------------------------
;; CALCULATE DELTA WEIGHT
;;  delta_weight = b * E * w
;; Note that the elements are vectors
(defun calculate_delta_weight (beta_shift error arc_weight)
   (vector_multiply3 beta_shift error arc_weight))

;;-------------------------------------------------
;; CALCULATE ERROR ONE HIDDEN NODE
;;-------------------------------------------------
;;
(defun calculate_final_error_one_hidden (incoming_error node_value)
   (* incoming_error (- 1 node_value) node_value))

;;-------------------------------------------------
;; CONVERT WEIGHT_LIST TO BACKPROP_WEIGHT LIST
;;  conv_weight_list
(defun conv_weight_list (x)
   (apply 'mapcar #'list x))


;;-------------------------------------------------
;; Does the usual forward evaluation, saving the values
;; of the hidden level
```

```
;;  conv_weight_list
(defun forward_eval (input w_list1 w_list2)
  (setf OUTPUT (level_eval (setf SAVED_HID
    (level_eval input w_list1)) w_list2))
)

;;-------------------------------------------------
;; Does a node evaluation in a back propagation
;;
(defun node_eval_back (inputs weigths)
        (summation (vector_multiply inputs weigths))
)

;;-------------------------------------------------
;; Does the level evaluation in the back propagation,
;; including the derivative
(defun level_eval_back (input_back w_list saved_level)
  (mapcar #'calculate_final_error_one_hidden
  (mapcar #'node_eval_back (make_input_list input_back w_list) w_list)
  saved_level)
  )

;;-------------------------------------------------
;; Creates a list of the beta values in the same
;; format of the weights list given

(defun list_beta (beta_shift w_list)
  (list_of (length w_list) (list_of (length(car w_list)) beta_shift)))

;;-------------------------------------------------
;; Function that applies calculate_delta_weight over
;; each element of the three lists.
;; Note that each element are also a list
(defun calc_delta_weight (beta_list error_list weight_list)
  (mapcar #'calculate_delta_weight beta_list error_list weight_list)
)

;;-------------------------------------------------
;; Calculates the new weights by just summing the
;; delta with the old weights

(defun calc_new_weight (delta_list w_list)
```

```
  (mapcar #'vector_sum delta_list w_list)
)

;;-------------------------------------------------
;; Function that calculates the new weigths
;;
(defun backeval (w_list error_list income_value input)
  (calc_new_weight
   (calc_delta_weight
    (list_beta beta w_list)
    (conv_weight_list(list_of (length input) error_list ))
    (list_of (length w_list) income_value))
   w_list
   )
)

;;-------------------------------------------------
;; output results
;;
(defun print_results (input output)
  (setf resultname (make-pathname :name "weilist.dat"))
  (setf outfile (open resultname :direction :output :if-exists :rename
      :if-does-not-exist :create))
  (format outfile "~%mean_error:~A~%" (mean_error SAVED_ERROR))
  (format outfile "~%Input:~A~%Exp_Output:~A~%Input Weights:~A~%Hidden Weights:~A"
  input output inp_wlist hid_wlist)
  (format outfile "~%CALC_OUTPUT:~A~%"OUTPUT_LIST)
  (close outfile)


  (setf resultname (make-pathname :name "weights.lisp"))
  (setf outfile (open resultname :direction :output :if-exists :rename
      :if-does-not-exist :create))
  (format outfile "~%(setf inp_wlist '~A)~%(setf hid_wlist '~A)~%"
  inp_wlist hid_wlist)
  (close outfile)

  'DONE
  )

;;-------------------------------------------------
;; function user to calculate the square root of the
```

```
;; mean of the errors
(defun inp_error (w_list)
  (sqrt(summation(vector_multiply w_list w_list))))

;;-------------------------------------------------
;; function user to calculate the mean error of several
;; inputs
(defun mean_error (w_list)
  (/ (summation (mapcar #'inp_error w_list))
     (length w_list))
)



;;-------------------------------------------------
;; function user to calculate the back prop of a
;; single input
;;
(defun calc_sing_inp (inp out)

      (setf inp_wlist
    (backeval inp_wlist (level_eval_back (calc_output_error out
(forward_eval inp inp_wlist hid_wlist))
(conv_weight_list hid_wlist) SAVED_HID) inp
inp)
    )
      (setf hid_wlist
    (backeval hid_wlist SAVED_ERROR_OUT SAVED_HID inp)
    )
      (setf SAVED_ERROR (append SAVED_ERROR (list SAVED_ERROR_OUT)))
      (setf OUTPUT_LIST (append OUTPUT_LIST (list OUTPUT)))
)

;;-------------------------------------------------
;; function for loading weights from previous training
;;
(defun load_weights (weights)
  (setf weightname (make-pathname :name weights))
  (with-open-file (str weightname :direction :input :if-does-not-exist :error)
  (do ((expression (read str nil 'eof)
   (read str nil 'eof)))
      ((eql expression 'eof))
```

```
    (eval expression)))
)



;;-------------------------------------------------
;; Main function
;; NOTE: This reuses the previous calculated weights
;; to the current single input
;;
(defun main (input output)
  (create_weights  input output)
;;  (load_weights "weights.lisp")
  (setf pathname (make-pathname :name "mean.txt"))
  (with-open-file (str pathname :direction :output :if-does-not-exist :create
       :if-exists :rename))
  (setf temp_error '2.0)
  (setf ERROR (mean_error SAVED_ERROR))
  (do (( i 1 (+ i 1)))
      (( < ERROR 0.02) (print_results input output))
      (setf SAVED_ERROR '())
      (setf OUTPUT_LIST '())
      (mapcar #'calc_sing_inp input output)
      (setf ERROR (mean_error SAVED_ERROR))
      (cond
       (( <  ERROR 0.01)(setf beta 0.005))
       (( <  ERROR 0.05)(setf beta 0.02))
       (( <  ERROR 0.1)(setf beta 0.025))
       (( <  ERROR 0.2)(setf beta 0.03))
       (( <  ERROR 0.3)(setf beta 0.045))
       (( <  ERROR 0.4)(setf beta 0.05))
       (( <  ERROR 0.5)(setf beta 0.055))
       (    t          (setf beta 0.06)))
      (setf temp_error ERROR)

      (if (zerop (rem i 10))
  (format t "~%mean_error:~A~%Iteration ~A~%beta:~A~%"
  (mean_error SAVED_ERROR) i beta))

      (if (zerop (rem i 200))
  (print_results input output))
```

```
    (with-open-file (str pathname :direction :output :if-exists :append
 :if-does-not-exist :create)
    (format str "~A ~A ~A~%" i (mean_error SAVED_ERROR)
    beta))
    )
)
```

APPENDIX B. PROLOG CODE OF A NEURAL NETWORK

```
/*****************************************************************/
/* Language    : Quintus Prolog Release 3.1.1                    */
/* Description: Program to implement a 3-layer NN with 15 inputs,  */
/*              15 hidden nodes and 5 output nodes.               */
/* Authors     : Prof. Neil Rowe and Jader Filho                 */
/* Last Update: 03/24/1997                                       */
/*****************************************************************/
assert_weights :- abolish(strength/3),abolish(strength/2),

  assertz(strength(hid_node,1,
[0.10280146,1.0383518,-0.15337943,13.348852,0.2140507,-4.999327,
11.041899,-1.376764,9.722739,-2.674495,7.643388,-0.11498318,
-0.74958926,1.1124623,1.2203351])),
  assertz(strength(hid_node,2,[1.3599663,2.0130012,-5.7742586,-6.144408,
-0.3617554,0.49077615,4.2990837,-1.3255733,-0.10300044,1.7859062,
-2.1418078,-0.8528033,-0.87857527,1.8577701,-0.6500677])),
  assertz(strength(hid_node,3,[7.299552,-0.11480039,4.1539264,-5.954579,
1.9724478,0.8563504,2.554235,3.1276588,-5.577703,-7.7033744,0.9631491,
-0.31658006,1.1658579,-0.7200168,-0.8751486])),
  assertz(strength(hid_node,4,[2.352219,0.9798964,7.387131,-4.302696,
1.3159721,-4.255448,-2.2222114,7.3860474,-0.110816374,-0.15095304,
1.3264731,1.761269,-1.9182163,-0.08970505,2.1381326])),
  assertz(strength(hid_node,5,[0.97137606,2.0978968,-1.5668068,5.3624883,
-0.3900549,0.7947839,-4.413633,1.5507306,-0.6049893,-0.5473221,
1.7741257,0.7199787,-1.7051593,1.9850469,0.3874647])),
  assertz(strength(hid_node,6,[-7.5278397,-3.1371894,6.8534527,9.540047,
3.971589,6.7969294,9.477654,-1.8458934,-13.109152,-2.2465706,
-15.705089,1.2939371,1.7863597,-0.2075032,0.7623969])),
  assertz(strength(hid_node,7,[-1.5873216,-1.033604,-1.7575526,
-11.586975,1.1679307,-2.6128724,2.420472,-0.14453241,1.3027858,
-2.2243912,9.150505,-0.76785827,0.19931579,-0.36563465,1.6281044])),
  assertz(strength(hid_node,8,[-11.580848,3.8350782,3.5109222,
4.163468,-3.4610188,-5.8272057,-4.719808,5.126142,3.3443942,
-0.88241,10.711175,2.0836523,0.43011373,0.7488813,-2.1782956])),
  assertz(strength(hid_node,9,[-11.183245,-2.8265104,-3.0190945,
3.825569,-9.887493,-8.926091,-8.578522,-9.81599,10.6942625,
5.7835407,1.2130237,1.3198781,-1.6814234,-0.7457888,-0.3643858])),
  assertz(strength(hid_node,10,[-9.208287,-0.5136238,2.4696598,1.4010974,
-3.0033565,-10.368849,-5.825708,8.918923,5.9908056,2.074206,
24.38145,0.3551642,-0.046705514,-0.271937,1.8054696])),
  assertz(strength(hid_node,11,[5.5495863,-1.3832068,15.012902,
-0.87178606,-1.5800465,2.5315452,-1.6559813,-9.013141,3.2857537,
```

```
-3.6133842,11.880634,-0.42965722,3.214854,0.019956164,-1.1296868])),
  assertz(strength(hid_node,12,[9.189349,9.68774,7.1437473,2.5421433,
-0.31797215,-1.2867705,0.6343578,5.5447206,4.213276,-8.133626,
-5.8887916,1.5713207,-0.4948193,-1.6689208,-0.98711646])),
  assertz(strength(hid_node,13,[11.598041,-0.612225,-9.021371,-11.469496,
1.5119609,0.075839214,1.7730261,-1.4520669,-5.056494,-0.42691928,
1.5322973,1.5731288,1.2850229,-0.81217283,0.45320952])),
  assertz(strength(hid_node,14,[3.8388643,1.3650446,-3.2947173,5.2161503,
-5.5149713,-5.4758544,-4.51567,-7.87467,3.2831373,1.8662292,
6.1908507,0.41444215,3.1471112,0.6712416,0.34379712])),
  assertz(strength(hid_node,15,[4.03495,0.12459024,-1.4378815,-13.7728405,
2.6996377,5.941888,7.072829,-8.149348,1.1662475,-3.6911345,8.328426,
-0.24888588,0.25269714,1.4399127,3.127232])),

  assertz(strength(out_node,[[6.545533,5.7997675,-1.7047756,-4.082592,
4.3749847,0.9555788,5.227637,1.7942159,-0.114222795,6.661545,
7.2471204,0.33012316,5.5946555,1.5604621,-12.4767685],[1.940118,
2.018785,-1.3316723,-1.7319521,3.1654818,10.293487,-14.334564,
8.87022,-8.368377,-12.34091,4.4616256,4.8475647,-2.2759478,
0.46862486,6.626953],[-6.262649,1.4892993,3.0227327,-0.91400874,
2.423398,-14.339465,-16.798841,14.285471,-7.921312,7.9008074,
4.1938386,10.859216,4.2084055,-5.5930195,1.3791995],[-8.259792,
-4.4346175,12.238324,5.9989915,2.793125,-10.320029,1.0851719,
-3.8516119,-7.3822327,-5.060156,0.2045436,-11.65261,-5.4408092,
13.733666,13.449633],[-2.0871668,-1.1725345,-10.751292,-7.295434,
-0.7270514,12.000224,-14.410152,0.3804165,0.47677583,4.5853095,
11.158862,-9.278686,7.908131,-2.4952629,6.78489]])).


:-ensure_loaded(library(math)),ensure_loaded(library(maplist)) ,
ensure_loaded(library(lists)), ensure_loaded('fig5.tmp'),
ensure_loaded('identity'),assert_weights.




go(OOO) :- tell(results),neural_net(O), maplist(nice_read,O,OO),
maplist(nicelist,OO,OOO), maplist(nicew,OOO),told.

nicew([K,X]) :- write('Shape '),writeq(K),write(' is: ('),
             maplist(nicewbit,X),write(' ) '),identity(X,ID),
     writeq(ID),!,nl.
```

```
nicew([K,X]).

nicewbit(X) :- write(' '),writeq(X).

nicelist([K,X],[K,NX]) :- maplist(conv_int,X,NX).

conv_int(X,NX) :- (X<0.5) -> NX is 0 | NX is 1.

neural_net(OO):- mid_level_out(ML), strength(out_node,W),length(W,Y),
   list_of(ML,Y,Z),maplist(neur,Z,W,YY),transpose(YY,OO).

mid_level_out(ML) :- bagof([SN,O],mid_level([SN,O]),ML).

mid_level1(IL,SL) :- input_list(hid_node,SN,IL),
   corresponding_strengths(hid_node,IL,SL).

mid_level([SN,O]) :- input_list(hid_node,SN,IL),
   corresponding_strengths(hid_node,IL,SL), neuron_eval(IL,SL,O).

inp_bag :- tell(out),bagof(IL,SN^input_list(hid_node,SN,IL),ML),
              maplist(lisp_input,ML),nl,fail.

lisp_input(X) :- write('('),maplist(lisp_input1,X),write(' )'),nl.
lisp_input1([K,X]) :- write(' '),writeq(X).

inp_bag :- told.

input_list(C,SN,IL) :- bagof([K,X],input(C,K,SN,X),IL).
input(hid_node,1,SN,X) :- shape(SN,A,_,_,_,_,_,_),sqrt(A,XX),X is XX/15.
input(hid_node,2,SN,X) :- shape(SN,_,C,B,_,_,_,_),X is B/(B+C).
input(hid_node,3,SN,X) :- shape(SN,_,_,_,[_,_,_,_,X,_],_,_,_).
input(hid_node,4,SN,X) :- shape(SN,_,_,_,[_,_,_,_,_,X],_,_,_).
input(hid_node,5,SN,X) :- shape(SN,_,_,_,_,[R,G,B|_],_,_),
   X is (R+G+B)/2000.
input(hid_node,6,SN,X) :- shape(SN,_,_,_,_,[R|_],_,_),
   X is R/650.
input(hid_node,7,SN,X) :- shape(SN,_,_,_,_,[_,G|_],_,_),
   X is G/650.
input(hid_node,8,SN,X) :- shape(SN,_,_,_,_,[_,_,B|_],_,_),
   X is B/650.
input(hid_node,9,SN,X) :- shape(SN,_,_,_,_,_,XX,_),X is XX/500.
input(hid_node,10,SN,X) :- shape(SN,_,_,_,_,[_,_,_,SR,SG,SB|_],_,_),
```

```
   X is (SR+SG+SB)/50.
input(hid_node,11,SN,X) :- shape(SN,A,_,_,_,_,_,_,[XX,_,_,_,_,_,_]),X is XX/A.
input(hid_node,12,SN,X) :- shape(SN,A,_,_,_,_,_,_,[_,XX,_,_,_,_,_]),X is XX/A.
input(hid_node,13,SN,X) :- shape(SN,A,_,_,_,_,_,_,[_,_,XX,_,_,_,_]),X is XX/A.
input(hid_node,14,SN,X) :- shape(SN,A,_,_,_,_,_,_,[_,_,_,XX,_,_,_]),X is XX/A.
input(hid_node,15,SN,X) :- shape(SN,A,_,_,_,_,_,_,[_,_,_,_,XX,_,_]),X is XX/A.


nice_read(I,[X,Y]) :- transpose(I,[[X|XX],Y]).

corresponding_strengths(C,[],[]).

corresponding_strengths(C,[[K,X]|IL],[S|SL]) :- strength(C,K,S),!,
   corresponding_strengths(C,IL,SL).


list_of([],L,[]).
list_of(M,0,[]).
list_of(M,1,[M]).
list_of(M,L,[M|ML]) :- LL is L-1,list_of(M,LL,ML),!.

member(X,L) :- append(_,[X|_],L).

neur([],_,[]).

neur([[K,X]|XX],Y,[[K,Z]|ZZ]):- maplist(mult,X,Y,Z1),sumup(Z1,SUM),
sigmoid(SUM,Z), !,neur(XX,Y,ZZ).

mult(X,Y,Z) :- Z is X*Y.

sumup([Item],Item).
sumup([Item|List],Sum) :- sumup(List,Partsum),Sum is Partsum+Item.

neuron_eval(IL,[],[]).

neuron_eval(IL,[EL|WL],[O|OO]) :- neuron(IL,EL,O), !, neuron_eval(IL,WL,OO).

neuron(IL,WL,O) :- node_eval(IL,WL,RO), sigmoid(RO,O).

node_eval([],[],0).
node_eval([[K,X]|IL],[S|SL],P) :- node_eval(IL,SL,P2), P is P2+(S*X).
```

```
node_eval([X|IL],[S|SL],P) :- node_eval(IL,SL,P2), P is P2+(S*X).

sigmoid(X,Y) :- XX is -1*X, exp(XX,YY),Y is 1/(1+YY).
```

APPENDIX C. REGIONS EXTRACTED OF AN IMAGE

Regions (shapes) extracted from Figure 7 by the Feature Extraction Program [Ref. 1]:

```
shape(1,278,45,50,[1,1,41,10,0.672,0.917],[848.655,853.058,842.784,48.28,
55.762,61.311,67.182,-0.554,0.17],105.344,[47.048,0.366,3.317,6,0,o]).
shape(2,493,117,23,[40,1,86,17,0.16,0.832],[663.862,610.069,582.138,51.039,
52.286,59.567,121.599,0.008,0.156],123.099,[94.213,0.37,4.62,11,1,o]).
shape(3,87,24,19,[53,1,71,8,0.094,0.94],[819.713,817.954,796.264,22.087,
25.83,36.713,66.173,-0.287,-0.095],148.086,[26.041,0.456,2.892,2,0,o]).
shape(4,5989,540,242,[1,1,110,88,0.058,0.053],[813.709,805.441,774.308,
50.457,67.9,73.765,89.904,0.045,0.34],155.262,[288.21,0.287,3.341,
36,9,o]).
shape(5,41,30,0,[90,3,104,6,0.752,0.908],[727.268,663.829,641.439,
29.686,41.122,41.645,76.388,0.261,0.321],101.607,[31.549,0.222,
0.555,2,0,c]).
shape(6,449,118,12,[1,4,45,22,0.602,0.719],[674.548,646.038,615.37,66.05,
71.78,82.613,133.459,-0.1,0.243],150.196,[119.912,0.349,4.376,16,0,o]).
shape(7,204,62,0,[58,26,80,39,0.244,0.276],[674.167,612.917,570.917,52.208,
54.834,62.344,119.918,0.269,-0.52],134.525,[63.808,0.361,3.09,10,1,c]).
shape(8,23,14,5,[1,32,7,36,0.95,0.234],[657.826,640.087,602.391,28.106,
32.336,39.417,94.349,0.058,0.02],110.229,[14.146,0.32,3.956,1,0,o]).
shape(9,101,39,5,[95,36,110,45,0.86,0.068],[693.129,626.475,582.545,41.357,
33.095,44.039,100.049,0.091,-0.188],104.579,[39.406,0.247,3.224,
3,1,o]).
shape(10,558,133,0,[39,37,66,76,0.02,0.251],[466.19,333.389,251.222,50.419,
62.019,63.829,115.373,0.332,0.245],237.639,[142.844,0.353,2.871,
18,1,c]).
shape(11,71,36,0,[66,39,74,50,0.252,0.016],[572.732,487.972,443.437,
44.348,52.718,59.354,115.552,0.224,-0.135],132.608,[37.621,0.343,
3.056,4,1,c]).
shape(12,101,53,0,[40,40,61,49,0.068,0.004],[667.04,598.366,541.812,
60.479,83.558,98.276,175.31,0.027,0.0],169.743,[55.908,0.369,
2.582,8,1,c]).
shape(13,115,47,0,[71,40,87,49,0.415,0.004],[630.035,563.513,519.609,
41.538,51.233,56.188,106.783,0.442,0.011],99.857,[48.964,0.298,
3.08,5,2,c]).
shape(14,69,38,0,[91,41,106,51,0.774,0.063],[722.377,652.159,615.203,
39.96,38.065,43.247,100.638,-0.149,-0.282],111.373,[41.211,0.429,
2.459,3,1,c]).
shape(15,57,45,0,[23,43,39,51,0.448,0.037],[678.509,608.719,547.877,36.067,
26.915,46.803,82.866,-0.443,-0.179],123.037,[48.034,0.391,3.925,10,
2,c]).
```

```
shape(16,48,35,0,[35,44,48,55,0.294,0.115],[580.125,484.5,420.75,45.856,
41.351,41.569,88.188,0.224,-0.426],104.671,[41.06,0.384,2.5,5,1,c]).
shape(17,18,10,6,[1,46,3,51,0.973,0.091],[646.0,589.333,524.167,50.047,
56.949,52.321,122.755,0.546,-0.417],147.224,[9.627,0.217,4.897,0,0,o]).
shape(18,92,42,0,[28,46,39,57,0.427,0.163],[619.761,546.587,484.5,44.445,
51.547,58.81,119.911,-0.1,-0.191],105.127,[42.662,0.332,2.745,4,2,c]).
shape(19,313,89,12,[47,63,76,88,0.059,0.731],[742.677,708.786,663.978,
72.554,103.744,111.509,141.487,-0.418,-0.413],221.277,[95.888,0.363,
3.307,10,0,o]).
shape(20,56,37,0,[61,65,66,81,0.164,0.634],[582.554,498.768,435.929,81.116,
99.608,116.033,206.514,0.213,0.217],232.437,[39.788,0.265,1.393,
2,1,c]).
shape(21,371,147,4,[20,68,59,88,0.285,0.794],[673.31,638.668,598.528,
61.291,76.491,89.743,138.089,0.164,0.109],145.861,[115.834,0.313,
4.12,14,4,o]).
shape(22,115,44,0,[24,76,39,86,0.445,0.856],[524.783,457.817,401.939,45.479,
51.306,63.537,103.258,0.268,0.082],151.837,[46.239,0.343,1.991,3,1,c]).
shape(23,31,17,14,[51,86,64,88,0.051,0.973],[567.032,494.645,438.71,29.46,
31.244,27.79,71.366,0.091,0.042],118.429,[17.152,0.278,3.793,0,0,o]).
```

APPENDIX D. C CODE FOR EXTRACTING WAVELET FEATURES AND SCANNING AN IMAGE

```
/*****************************************************************************
 File: trainset.c
 Name: Jader Gomes da Silva Filho
 Date: 05/97
 Operating Environment: Windows 95
 Compiler: MS Visual C++
 Description: Program used to extract features from a image
*****************************************************************************/

/*.....Include necessary definitions  */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mil.h>
#include <math.h>
#include "wtn.h"
#include "daub4.h"
#include "nrutil.h"
#include "util.h"

void main(void)
{
  MIL_ID MilApplication;  /* Application identifier.  */
  MIL_ID MilSystem;       /* System identifier.       */
  MIL_ID MilGraphId;      /* Graphics context identifier. */
  MIL_ID MilDisplay0;     /* Display 0 identifier.    */
  MIL_ID MilDisplay1;     /* Display 1 identifier     */
  MIL_ID MilDisplay2;     /* Display 2 identifier     */
  MIL_ID MilDisplay3;     /* Display 3 identifier     */
  MIL_ID MilDisplay4;     /* Display 4 identifier     */
  MIL_ID Disp_Level_4;    /* Display level_4 identifier. */
  MIL_ID MilImage0;       /* Image 0 buffer identifier. */
  MIL_ID MilImage1;       /* Image 1 buffer identifier. */
  MIL_ID MilImage2;       /* Image 2 buffer identifier. */
  MIL_ID MilImage3;       /* Image 3 buffer identifier. */
  MIL_ID MilImage4;       /* Image 4 buffer identifier. */
  MIL_ID Level_4;         /* Image level_4  buffer identifier. */
  long   ImageSizeX=IMAGE_WIDTH;  /* Image width. */
  long   ImageSizeY=IMAGE_HEIGHT; /* Image height.*/
  unsigned char  *tiff_data;      /* Tiff image  */
  unsigned char  *init_data;      /* Tiff image  */
   /* unsigned char  *back_data;     Tiff image  */
```

```
unsigned char  *mod_tiff_d;      /* Tiff image  */
double          *wavelets;       /* Wavelets     */
double          *wbuf;           /* Tmp wavelet buffer */
double          *level_wbuf;     /* Level4 of wavelets */
char           *Image_file; /* Name of Image file */
char            Image_list[STRING] = "newone1.txt";
char            training_file[STRING] = "result1/training";
char            id_file[STRING] = "result1/id";
char           *name;
unsigned long  dim[2]={IMAGE_WIDTH, IMAGE_HEIGHT};
int            i,j,elim=0;
double          max,min,factor,limit;
char           *Wave_file = "result1/waveout.tif";
int            h_level_size = IMAGE_WIDTH/(pow(2,LEVEL));
int            v_level_size = IMAGE_HEIGHT/(pow(2,LEVEL));
int            indice;
unsigned long  dim_w[2]={h_level_size,v_level_size}; /* for level4 inverse transf
Feature_Node   *output_level;
unsigned char  *level_tiff_d;
int            wlevel = LEVEL;
int            angle = 0, step_angle = STEP_ANGLE;
double         level_max, level_min;
Training_List *names, *first_file;
int k = 0, num_files;

/*
 *
 *.....Initialize MIL
 *
 */
/* Allocate Arrays */
Image_file = (char *) malloc(STRING*sizeof(char));
name       = (char *) malloc(STRING*sizeof(char));

init_data = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
tiff_data = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
mod_tiff_d = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
wavelets  = (double *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND*sizeof(double));
```

```
wbuf  = (double *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND*sizeof(double));
output_level = (Feature_Node *)
  malloc(SIZE_LIST*sizeof(Feature_Node));


if ((init_data == NULL) || (tiff_data == NULL) || (mod_tiff_d == NULL) ||
    (wavelets == NULL) || (wbuf == NULL) || (output_level == NULL)){
  printf("Out of memory\n");
  exit(0);
}
/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem,
 &MilDisplay0,M_NULL, M_NULL);
MdispAlloc(MilSystem,M_DEV1,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay1);
MdispAlloc(MilSystem,M_DEV2,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay2);
MdispAlloc(MilSystem,M_DEV3,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay3);
MdispAlloc(MilSystem,M_DEV4,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay4);
MdispAlloc(MilSystem,M_DEV6,M_DISPLAY_SETUP,M_DEFAULT,
           &Disp_Level_4);
/* Find the best size for the display image depending on the display type. *
if (MdispInquire(MilDisplay0,M_DISP_MODE,M_NULL)==M_WINDOWED)
  {
    /* Smallest possible. */
    ImageSizeX = IMAGE_WIDTH;
    ImageSizeY = IMAGE_HEIGHT;
  }else{
    /* The size of the entire display to avoid possible display artifact. */
    ImageSizeX = min(MdispInquire(MilDisplay0,M_SIZE_X,M_NULL),
     M_DEF_IMAGE_SIZE_X_MAX);
    ImageSizeY = min(MdispInquire(MilDisplay0,M_SIZE_Y,M_NULL),
     M_DEF_IMAGE_SIZE_Y_MAX);
  }
/* Allocate a two-dimensional image buffer to perform graphics in it. */
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
```

```
    &MilImage0);
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage1);
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP,
    &MilImage2);
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage3);
MbufAlloc2d(MilSystem,
    ACTUAL_WIDTH,
    ACTUAL_HEIGHT,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage4);
MbufAlloc2d(MilSystem,
    h_level_size,
    v_level_size,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP,
    &Level_4);

/* Enable keying on display if its supported. */
if ((M_DEF_DISPLAY_KEY_ENABLE_ON_ALLOC != 0) &&
    MdispInquire(MilDisplay0,M_DISP_KEY_SUPPORTED,0))
  MdispOverlayKey(MilDisplay0,M_KEY_ON_COLOR,M_EQUAL,
                  0xFFL,M_DEF_DISPLAY_KEY_COLOR);

/***.....Load original image**/
num_files = load_names_and_types(Image_list, &first_file);
names = first_file;

for(wlevel=LEVEL; wlevel<LEVEL+LEVEL_NIVEL;wlevel++){

  h_level_size = IMAGE_WIDTH/(pow(2,wlevel));
  v_level_size = IMAGE_HEIGHT/(pow(2,wlevel));
  dim_w[0] = h_level_size;
```

```
dim_w[1] = v_level_size;

level_wbuf  = (double *)
  malloc(h_level_size*v_level_size*IMAGE_BAND*sizeof(double));
level_tiff_d  = (unsigned char *)
  malloc(h_level_size*v_level_size*IMAGE_BAND);
if ((level_wbuf == NULL) || (level_tiff_d == NULL)){
  printf("Out of memory\n");
  exit(0);
}

printf("\nProcessing level %d.\n", wlevel);

init_write_lisp_training(training_file,wlevel);

do {

  strcpy(Image_file,names->file_name);
  strcpy(name,Image_file);
  strcat(Image_file,".tif");
  printf("\n%s\n",name);

  /* Clear Buffers   */
  MbufClear(MilImage0,BACK);
  MbufClear(MilImage1,BACK);
  MbufClear(MilImage2,BACK);
  MbufClear(MilImage3,BACK);
  MbufClear(MilImage4,BACK);
  MbufClear(Level_4,BACK);


  /* Load the images from the file. */
  MbufImport(Image_file, M_TIFF, M_LOAD, MilSystem, &MilImage4);

  MimTranslate(MilImage4, MilImage0, (IMAGE_WIDTH - ACTUAL_WIDTH)/2,
(IMAGE_HEIGHT - ACTUAL_HEIGHT)/2, M_DEFAULT);

  MdispSelect(MilDisplay0, MilImage0);
  printf("\nImage %d displayed.\n", k+1);


  /*.....Compute wavelet transform*/
```

```
      for(angle = 0; angle < 360; angle += step_angle){
printf("\nProcessing angle %d.\n", angle);

MbufGet(MilImage0,tiff_data);

copy_char_to_double(tiff_data,wavelets);
log_of(wavelets);
wtn(wavelets,dim,2,1,daub4);
max =  min = wavelets[0];
for(i=0;i < IMAGE_WIDTH; i++)
  for(j=0; j < IMAGE_HEIGHT; j++){
    if(wavelets[j*IMAGE_WIDTH+i] > max)
      max = wavelets[j*IMAGE_WIDTH+i];
    if(wavelets[j*IMAGE_WIDTH+i] < min)
      min = wavelets[j*IMAGE_WIDTH+i];
  }

printf("Max wavelet Coeff = %f\n",max);
printf("Min wavelet Coeff = %f\n",min);
if(fabs(max) > fabs(min)) limit = fabs(max);
else limit = fabs(min);
/* Display wavelets */
factor = (double)0xffff/limit;
copy_double_to_char_rev(wavelets,mod_tiff_d,factor);

display_buffer_on_screen(&MilDisplay1,&MilImage1,
 mod_tiff_d,"\nWavelets displayed.",0);

/*recovering the original picture with the inverse DWT*/
copy_double_to_double(wavelets,wbuf);
wtn(wbuf,dim,2,-1,daub4);
copy_double_to_char(wbuf,mod_tiff_d,1.0);
display_buffer_on_screen(&MilDisplay2,&MilImage2,mod_tiff_d,
 "\nRestored Image displayed.",0);
MgraAlloc(MilSystem,&MilGraphId);
MgraColor(MilGraphId,0);
/* Box around current level */
MgraRect(MilGraphId,MilImage1,h_level_size,v_level_size,
 2*h_level_size-1,2*v_level_size-1);
MdispSelect(MilDisplay1, MilImage1);

/* building the level buffer*/
```

```
level_max = level_min = fabs(wavelets[v_level_size*
     IMAGE_WIDTH+h_level_size]);
for(i=h_level_size;i < h_level_size*2; i++)
  for(j=v_level_size;j < v_level_size*2; j++){
    indice = (j-v_level_size)*h_level_size+i-h_level_size;
    level_wbuf[indice] = wavelets[j*IMAGE_WIDTH+i];
    if(fabs(wavelets[j*IMAGE_WIDTH+i]) > level_max)
      level_max = fabs(wavelets[j*IMAGE_WIDTH+i]);
    if(fabs(wavelets[j*IMAGE_WIDTH+i]) < level_min)
      level_min = fabs(wavelets[j*IMAGE_WIDTH+i]);
  }
printf("Max level_wavelet Coeff = %f\n",level_max);
printf("Min level_wavelet Coeff = %f\n",level_min);

normalize(level_wbuf, level_max/MAX_VALUE, h_level_size, v_level_size);

write_wav_level_to_disk(level_wbuf, name, wlevel, angle,
h_level_size, v_level_size);

copy_double_to_char_rev_level(level_wbuf,level_tiff_d,factor,
h_level_size, v_level_size);
find_list_highest(level_wbuf, output_level,
  h_level_size, v_level_size);
printf("\n");
for (i = 0; i < SIZE_LIST ; i++){
  printf("\nx = %5.3f, y = %5.3f, value = %5.3f, mean = %5.3f, variance = %5.3
 output_level[i].point.x, output_level[i].point.y,
 output_level[i].value, output_level[i].mean,
 output_level[i].variance);
}
printf("\n");
write_wav_level_high_to_disk(output_level, name, wlevel, angle,
     h_level_size, v_level_size);

write_lisp_training(output_level, name, wlevel, angle, training_file);

MbufExport(Wave_file, M_TIFF, MilImage1);
MbufClear(MilImage0,BACK);
MimRotate(MilImage4, MilImage0, angle+step_angle, M_DEFAULT,
  M_DEFAULT, M_DEFAULT, M_DEFAULT,M_BICUBIC);
MdispSelect(MilDisplay0, MilImage0);
      }
```

```
      k++;
    }while((names = names->next) != NULL);
    free(level_wbuf);
    free(level_tiff_d);

    end_write_lisp_training(training_file, num_files, first_file, wlevel);
    write_lisp_id_file(id_file, num_files, first_file, wlevel);
  }

  /*.....Clean Up*/
  /* Disable keying on display if its supported. */
  if ((M_DEF_DISPLAY_KEY_DISABLE_ON_FREE != 0) &&
      MdispInquire(MilDisplay0,M_DISP_KEY_SUPPORTED,0))
    MdispOverlayKey(MilDisplay0,M_KEY_OFF,M_NULL,M_NULL,M_NULL);
  /* Release subimages and color image buffer. */
  MbufFree(MilImage0);
  MbufFree(MilImage1);
  MbufFree(MilImage2);
  MbufFree(MilImage3);
  MbufFree(MilImage4);
  MbufFree(Level_4);
  /* Release defaults. */
  MdispFree(MilDisplay1);
  MdispFree(MilDisplay2);
  MdispFree(MilDisplay3);
  MdispFree(MilDisplay4);
  MdispFree(Disp_Level_4);
  MappFreeDefault(MilApplication, MilSystem, MilDisplay0, M_NULL, M_NULL);
  free(output_level);
  free_file_list(&first_file);
  return;
}
```

```
/*****************************************************************************
 File: scanning.c
 Name: Jader Gomes da Silva Filho
 Date: 05/97
 Operating Environment: Windows 95
 Compiler: MS Visual C++
 Description: Scans an image generating a file with a set of feature vectors
*****************************************************************************/
 *.....Include necessary definitions  */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mil.h>
#include <math.h>
#include "wtn.h"
#include "daub4.h"
#include "nrutil.h"
#include "util.h"

void main(void)
{
  MIL_ID MilApplication;  /* Application identifier.  */
  MIL_ID MilSystem;       /* System identifier.       */
  MIL_ID MilGraphId;      /* Graphics context identifier. */
  MIL_ID MilDisplay0;     /* Display 0 identifier.    */
  MIL_ID MilDisplay1;     /* Display 1 identifier     */
  MIL_ID MilDisplay2;     /* Display 2 identifier     */
  MIL_ID MilDisplay3;     /* Display 3 identifier     */
  MIL_ID MilDisplay4;     /* Display 4 identifier     */
  MIL_ID Disp_Level_4;    /* Display level_4 identifier. */
  MIL_ID MilImage0;       /* Image 0 buffer identifier. */
  MIL_ID MilImage1;       /* Image 1 buffer identifier. */
  MIL_ID MilImage2;       /* Image 2 buffer identifier. */
  MIL_ID MilImage3;       /* Image 3 buffer identifier. */
  MIL_ID MilImage4;       /* Image 4 buffer identifier. */
   MIL_ID Level_4;         /* Image level_4  buffer identifier. */
  long   ImageSizeX=IMAGE_WIDTH;  /* Image width. */
  long   ImageSizeY=IMAGE_HEIGHT; /* Image height.*/
  unsigned char  *tiff_data;      /* Tiff image  */
  unsigned char  *init_data;      /* Tiff image  */
   /* unsigned char  *back_data;     Tiff image  */
  unsigned char  *mod_tiff_d;      /* Tiff image  */
```

```
double          *wavelets;       /* Wavelets     */
double          *wbuf;           /* Tmp wavelet buffer */
double          *level_wbuf;     /* Level4 of wavelets */
char           *Image_file; /* Name of Image file */
char            Image_list[STRING] = "newone1.txt";
char            test_file[STRING] = "result3/scan";
char            win_id[STRING] = "result3/win_id";
char           *name;
unsigned long  dim[2]={IMAGE_WIDTH, IMAGE_HEIGHT};
int            i,j,elim=0;
double          max,min,factor,limit;
char           *Wave_file = "result3/waveout3.tif";
int            h_level_size = IMAGE_WIDTH/(pow(2,LEVEL));
int            v_level_size = IMAGE_HEIGHT/(pow(2,LEVEL));
int            indice;
unsigned long  dim_w[2]={h_level_size,v_level_size}; /* for level4 inverse trans
Feature_Node   *output_level;
unsigned char  *level_tiff_d;
int            wlevel = LEVEL;
double         level_max, level_min;
char *names;
int k = 0, num_files;

/*
 *
 *.....Initialize MIL
 *
 */
/* Allocate Arrays */
Image_file = (char *) malloc(STRING*sizeof(char));
name       = (char *) malloc(STRING*sizeof(char));

init_data = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
tiff_data = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
mod_tiff_d = (unsigned char *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND);
wavelets  = (double *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND*sizeof(double));
wbuf  = (double *)
  malloc(ImageSizeX*ImageSizeY*IMAGE_BAND*sizeof(double));
```

```
output_level = (Feature_Node *)
  malloc(SIZE_LIST*sizeof(Feature_Node));


if ((init_data == NULL) || (tiff_data == NULL) || (mod_tiff_d == NULL) ||
    (wavelets == NULL) || (wbuf == NULL) || (output_level == NULL)){
  printf("Out of memory\n");
  exit(0);
}
/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem,
 &MilDisplay0,M_NULL, M_NULL);
MdispAlloc(MilSystem,M_DEV1,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay1);
MdispAlloc(MilSystem,M_DEV2,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay2);
MdispAlloc(MilSystem,M_DEV3,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay3);
MdispAlloc(MilSystem,M_DEV4,M_DISPLAY_SETUP,M_DEFAULT,
           &MilDisplay4);
MdispAlloc(MilSystem,M_DEV6,M_DISPLAY_SETUP,M_DEFAULT,
           &Disp_Level_4);
/* Find the best size for the display image depending on the display type. *
if (MdispInquire(MilDisplay0,M_DISP_MODE,M_NULL)==M_WINDOWED)
  {
    /* Smallest possible. */
    ImageSizeX = IMAGE_WIDTH;
    ImageSizeY = IMAGE_HEIGHT;
  }else{
    /* The size of the entire display to avoid possible display artifact. */
    ImageSizeX = min(MdispInquire(MilDisplay0,M_SIZE_X,M_NULL),
     M_DEF_IMAGE_SIZE_X_MAX);
    ImageSizeY = min(MdispInquire(MilDisplay0,M_SIZE_Y,M_NULL),
     M_DEF_IMAGE_SIZE_Y_MAX);
  }
/* Allocate a two-dimensional image buffer to perform graphics in it. */
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage0);
MbufAlloc2d(MilSystem,
```

```
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage1);
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP,
    &MilImage2);
MbufAlloc2d(MilSystem,
    ImageSizeX,
    ImageSizeY,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage3);
MbufAlloc2d(MilSystem,
    ACTUAL_WIDTH,
    ACTUAL_HEIGHT,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP+M_PROC,
    &MilImage4);
MbufAlloc2d(MilSystem,
    h_level_size,
    v_level_size,
    IMAGE_DEPTH+M_UNSIGNED, M_IMAGE+M_DISP,
    &Level_4);

/* Enable keying on display if its supported. */
if ((M_DEF_DISPLAY_KEY_ENABLE_ON_ALLOC != 0) &&
    MdispInquire(MilDisplay0,M_DISP_KEY_SUPPORTED,0))
  MdispOverlayKey(MilDisplay0,M_KEY_ON_COLOR,M_EQUAL,
                  0xFFL,M_DEF_DISPLAY_KEY_COLOR);

num_files = load_names(Image_list, &names);

for(wlevel=LEVEL; wlevel<LEVEL+LEVEL_NIVEL;wlevel++){

  h_level_size = IMAGE_WIDTH/(pow(2,wlevel));
  v_level_size = IMAGE_HEIGHT/(pow(2,wlevel));
  dim_w[0] = h_level_size;
  dim_w[1] = v_level_size;

  level_wbuf  = (double *)
    malloc(h_level_size*v_level_size*IMAGE_BAND*sizeof(double));
```

```
level_tiff_d  = (unsigned char *)
  malloc(h_level_size*v_level_size*IMAGE_BAND);
if ((level_wbuf == NULL) || (level_tiff_d == NULL)){
  printf("Out of memory\n");
  exit(0);
}

printf("\nProcessing level %d.\n", wlevel);

/*init_write_lisp_test(test_file,wlevel);*/

for(k=0;k<num_files;k++){

  strcpy(Image_file,names+k*STRING);
  strcpy(name,Image_file);
  strcat(Image_file,".tif");
  printf("\n%s\n",name);

  /* Clear Buffers   */
  MbufClear(MilImage0,BACK);
  MbufClear(MilImage1,BACK);
  MbufClear(MilImage2,BACK);
  MbufClear(MilImage3,BACK);
  MbufClear(MilImage4,BACK);
  MbufClear(Level_4,BACK);


  /* Load the images from the file. */
  MbufImport(Image_file, M_TIFF, M_LOAD, MilSystem, &MilImage4);

  /* Translate the original image . */

  MimTranslate(MilImage4, MilImage0, (IMAGE_WIDTH - ACTUAL_WIDTH)/2,
(IMAGE_HEIGHT - ACTUAL_HEIGHT)/2, M_DEFAULT);

  MdispSelect(MilDisplay0, MilImage0);
  printf("\nImage %d displayed.\n", k+1);

  /*
   *
   *.....Compute wavelet transform
   *
```

```
       */

      MbufGet(MilImage0,tiff_data);

      copy_char_to_double(tiff_data,wavelets);
      log_of(wavelets);
      wtn(wavelets,dim,2,1,daub4);
      max =  min = wavelets[0];
      for(i=0;i < IMAGE_WIDTH; i++)
for(j=0; j < IMAGE_HEIGHT; j++){
  if(wavelets[j*IMAGE_WIDTH+i] > max)
    max = wavelets[j*IMAGE_WIDTH+i];
  if(wavelets[j*IMAGE_WIDTH+i] < min)
    min = wavelets[j*IMAGE_WIDTH+i];
}

      printf("Max wavelet Coeff = %f\n",max);
      printf("Min wavelet Coeff = %f\n",min);
      if(fabs(max) > fabs(min)) limit = fabs(max);
      else limit = fabs(min);
      /* Display wavelets */
      factor = (double)0xffff/limit;
      copy_double_to_char_rev(wavelets,mod_tiff_d,factor);

      display_buffer_on_screen(&MilDisplay1,&MilImage1,
       mod_tiff_d,"\nWavelets displayed.",0);

      /*recovering the original picture with the inverse DWT*/
      copy_double_to_double(wavelets,wbuf);
      wtn(wbuf,dim,2,-1,daub4);
      copy_double_to_char(wbuf,mod_tiff_d,1.0);
      /*    display_buffer_on_screen(&MilDisplay2,&MilImage2,
    mod_tiff_d,"\nRestored Image displayed.",0);*/

      MgraAlloc(MilSystem,&MilGraphId);
      MgraColor(MilGraphId,0);
      /* Box around current level */
      MgraRect(MilGraphId,MilImage1,h_level_size,v_level_size,
       2*h_level_size-1,2*v_level_size-1);
      /* Box around level 1
 MgraRect(MilGraphId,MilImage1,IMAGE_WIDTH/2,IMAGE_HEIGHT/2,
 IMAGE_WIDTH-1,IMAGE_HEIGHT-1);*/
```

```
      MdispSelect(MilDisplay1, MilImage1);

      /* building the level buffer*/
      level_max = level_min = fabs(wavelets[v_level_size*
   IMAGE_WIDTH+h_level_size]);
      for(i=h_level_size;i < h_level_size*2; i++)
for(j=v_level_size;j < v_level_size*2; j++){
  indice = (j-v_level_size)*h_level_size+i-h_level_size;
  level_wbuf[indice] = wavelets[j*IMAGE_WIDTH+i];
  if(fabs(wavelets[j*IMAGE_WIDTH+i]) > level_max)
    level_max = fabs(wavelets[j*IMAGE_WIDTH+i]);
  if(fabs(wavelets[j*IMAGE_WIDTH+i]) < level_min)
    level_min = fabs(wavelets[j*IMAGE_WIDTH+i]);
}
      printf("Max level_wavelet Coeff = %f\n",level_max);
      printf("Min level_wavelet Coeff = %f\n",level_min);

      normalize(level_wbuf, level_max/HIST_VALUE, h_level_size, v_level_size);

      histogram(level_wbuf, wlevel, HIST_VALUE);

      normalize(level_wbuf, HIST_VALUE/MAX_VALUE, h_level_size, v_level_size);

      write_wav_level_to_disk(level_wbuf, name, wlevel, 2,
      h_level_size, v_level_size);

      copy_double_to_char_rev_level(level_wbuf,level_tiff_d,factor,
   h_level_size, v_level_size);
      /* MbufPut(Level_4,level_tiff_d);
 display_buffer_on_screen(&Disp_Level_4,&Level_4,
 level_tiff_d,"\nLevel_Wbuf displayed.",0);
 */
      find_list_highest(level_wbuf, output_level, h_level_size,
v_level_size);

      for (i = 0; i < SIZE_LIST ; i++){
printf("\nx = %5.3f, y = %5.3f, value = %5.3f, mean = %5.3f, variance = %5.3f"
       output_level[i].point.x, output_level[i].point.y,
       output_level[i].value, output_level[i].mean,
       output_level[i].variance);
      }
      printf("\n");
```

```
      write_wav_level_high_to_disk(output_level, name, wlevel, 2,
   h_level_size, v_level_size);

      /*write_lisp_test(output_level, name, wlevel, test_file);*/
      scan_picture(level_wbuf, output_level, wlevel, h_level_size,
  v_level_size, name, win_id);
      MbufExport(Wave_file, M_TIFF, MilImage1);
    }
    free(level_wbuf);
    free(level_tiff_d);

    /*end_write_lisp_test(test_file, wlevel);*/
  }

  /* Clean Up */
  /* Disable keying on display if its supported. */
  if ((M_DEF_DISPLAY_KEY_DISABLE_ON_FREE != 0) &&
      MdispInquire(MilDisplay0,M_DISP_KEY_SUPPORTED,0))
    MdispOverlayKey(MilDisplay0,M_KEY_OFF,M_NULL,M_NULL,M_NULL);
  /* Release subimages and color image buffer. */
  MbufFree(MilImage0);
  MbufFree(MilImage1);
  MbufFree(MilImage2);
  MbufFree(MilImage3);
  MbufFree(MilImage4);
  MbufFree(Level_4);
  /* Release defaults. */
  MdispFree(MilDisplay1);
  MdispFree(MilDisplay2);
  MdispFree(MilDisplay3);
  MdispFree(MilDisplay4);
  MdispFree(Disp_Level_4);
  MappFreeDefault(MilApplication, MilSystem, MilDisplay0, M_NULL, M_NULL);
  free(output_level);
  free(names);
  return;
}
```

```
/*****************************************************************************
 File: util.h
 Name: Jader Gomes da Silva Filho
 Date: 05/97
 Operating Environment: Windows 95
 Compiler: MS Visual C++
*****************************************************************************/

#include "mil.h"

#define IMAGE_WIDTH          512L
#define IMAGE_HEIGHT         512L
#define ACTUAL_WIDTH         512L
#define ACTUAL_HEIGHT        512L
#define IMAGE_DEPTH    8L
#define IMAGE_BAND    1L
#define SIZE_LIST     10
#define STEP_ANGLE   360
#define LEVEL    3
#define LEVEL_NIVEL    1
#define SCAN_WIN    2
#define SCAN_STEP   2
#define SWAP(x,y) {int t; t = x; x = y; y = t;}
#define FALSE    0
#define TRUE     1
#define STRING 256
#define BACK   128L
#define MAX_VALUE   10.0F
#define HIST_VALUE   4000.0F

typedef struct point {
  double x,y;
} Point;

typedef struct feature_node {
  Point point;
  double value;
  double mean;
  double variance;
} Feature_Node;

typedef struct training_list{
```

```
  char *file_name;
  int  same_type;
  struct training_list
    *next;
} Training_List;

void write_wavelets_to_disk(double *coef,char *name);

void write_message_and_pause(char *message);

void display_buffer_on_screen(MIL_ID *Display, MIL_ID *Image,
                              unsigned char *data, char *message,
                              int wait);

void copy_double_to_double(double *in, double *out);

void copy_double_to_double_level(double *in, double *out, int h_size, int v_size);

void copy_double_to_char_rev(double *in, unsigned char *out,double scale);

void copy_double_to_char_rev_level(double *in, unsigned char *out,double scale,
     int h_size, int v_size);

void copy_double_to_char(double *in, unsigned char *out,double scale);

void copy_double_to_char_level(double *in, unsigned char *out,double scale,
 int h_size, int v_size);

void copy_char_to_double(unsigned char *in, double *out);

void find_list_highest(double *in, Feature_Node *out, int h_size, int v_size);

void write_wav_level_to_disk(double *in, char *name, int level, int angle,
     int h_size, int v_size);
void write_wav_level_high_to_disk(Feature_Node *feature_list, char *name,
     int level, int angle, int h_size, int v_size);
void find_list_lowest(double *in, Feature_Node *feature_list,
       int h_size, int v_size);
void log_of(double *inout);

Point calc_centroid(Feature_Node *feature_list, int level);
```

```
int load_names_and_types(char *file_list,Training_List **names_list);

void normalize(double *inout, double factor, int h_size, int v_size);

void init_write_lisp_training(char *file, int level);

void write_lisp_training(Feature_Node *feature_list, char *name,
 int level, int angle, char *file);

void end_write_lisp_training(char *file, int num_files, Training_List *names,
     int level);

void write_lisp_id_file(char *file, int num_files, Training_List *names,
int level);

int load_names(char *file_list, char **names);

void free_file_list(Training_List **names_list);

int Int(double value);

/*************************************************************************/
/*              Phase 2 - Testing                                        */
/*************************************************************************/

void init_write_lisp_test(char *file, int level);
void write_lisp_test(Feature_Node *feature_list, char *name,
 int level, char *file);
void end_write_lisp_test(char *file, int level);

/*************************************************************************/
/*              Phase 3 - Scanning                                       */
/*************************************************************************/

void histogram(double *in, int level, double level_max);

void scan_picture(double *in, Feature_Node *out, int level, int h_level_size,
  int v_level_size, char *name, char *win_id);
```

```
/*****************************************************************************
 File: util.c
 Name: Jader Gomes da Silva Filho
 Date: 05/97
 Operating Environment: Windows 95
 Compiler: MS Visual C++
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mil.h>
#include <math.h> /* for fabs */
#include "util.h"
#include "nrutil.h"


/* Output for Matlab Plotting*/
void write_wav_level_high_to_disk(Feature_Node *feature_list, char *name,
     int level, int angle, int h_size, int v_size)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];
  int i, temp_x, temp_y, max_index = SIZE_LIST;
  Point centroid = calc_centroid(feature_list, level);

  strcpy(file_n1,name);
  /*sprintf(file_n2,"l%da%d",level,angle); */
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".m");
  outfile = fopen(file_n1,"a");
  if(outfile == (FILE *)-1){
    printf("\nError writing wavelets to files %s\n",file_n1);
    exit(0);
  }
  else{
    fprintf(outfile,"%%Highest %d Level %d Angle %d Figure %s\n",
    max_index,level,angle,name);
    fprintf(outfile,"xh%d%d = 0:1:%d;\n",level,angle,h_size-1);
    fprintf(outfile,"yh%d%d = 0:1:%d;\n",level,angle,v_size-1);
    fprintf(outfile,"value%d%d = zeros(%d,%d);\n",level,angle,v_size,h_size);
```

```
    fprintf(outfile,"mean%d%d = zeros(%d,%d);\n",level,angle,v_size,h_size);
    fprintf(outfile,"variance%d%d = zeros(%d,%d);\n",level,angle,v_size,h_size)
    for(i=0; i < max_index; i++)
      {
temp_x = feature_list[i].point.x + 1;
temp_y = feature_list[i].point.y + 1;
/*printf("\nx = %d y = %d", temp_x, temp_y);*/
fprintf(outfile,"value%d%d(%d,%d) = %.6lf;\n",level,angle,temp_y,temp_x,
feature_list[i].value);
fprintf(outfile,"mean%d%d(%d,%d) = %.6lf;\n",level,angle,temp_y,temp_x,
feature_list[i].mean);
fprintf(outfile,"variance%d%d(%d,%d) = %.6lf;\n",level,angle,temp_y,temp_x,
feature_list[i].variance);
      }
    fprintf(outfile, "xh%d%d = xh%d%d - %.6lf;\n",level,angle,
    level,angle,centroid.x);
    fprintf(outfile, "yh%d%d = yh%d%d - %.6lf;\n",level,angle,
    level,angle,centroid.y);
    fprintf(outfile,"figure(2)\n");
    fprintf(outfile,"clf\n");
    fprintf(outfile,"surf(xh%d%d,yh%d%d,value%d%d),shading interp, view(2), col
    level,angle);
    fprintf(outfile,"set(gca,'YDir','reverse')\n");
    fprintf(outfile,"title('Highest %d Level %d Angle %d Figure %s')\n",
    max_index,level,angle,name);
    fprintf(outfile,"title('Highest %d Level %d Figure %s')\n",
    max_index,level,name);
    fprintf(outfile,"xlabel('x')\n");
    fprintf(outfile,"ylabel('y')\n");
    fprintf(outfile,"zlabel('coef')\n");
    fprintf(outfile,"figure(3)\n");
    fprintf(outfile,"clf\n");
    fprintf(outfile,"surf(xh%d%d,yh%d%d,abs(value%d%d)),shading interp, view(2)
    level,angle,level,angle);
    fprintf(outfile,"set(gca,'YDir','reverse')\n");
    fprintf(outfile,"title('Highest %d Level %d Angle %d Figure %s')\n",
    max_index,level,angle,name);
    fprintf(outfile,"title('Highest %d Level %d Figure %s')\n",
    max_index,level,name);
    fprintf(outfile,"xlabel('x')\n");
    fprintf(outfile,"ylabel('y')\n");
    fprintf(outfile,"zlabel('abs(coef)')\n");
```

```
  }
  fclose(outfile);
  return;
}
/* Output for Matlab Plotting*/
void write_wav_level_to_disk(double *in, char *name, int level, int angle,
     int h_size, int v_size)
{
  FILE *outfile;
  char file_n1[STRING],file_n2[STRING];
  int i,j,indice;

  strcpy(file_n1,name);
  /*sprintf(file_n2,"l%da%d",level,angle); */
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".m");
  outfile = fopen(file_n1,"w");
  if(outfile == (FILE *)-1){
    printf("\nError writing wavelets to files %s\n",file_n1);
    exit(0);
  }
  else{
    fprintf(outfile,"%%Wavelets level %d, angle %d, figure %s\n",
    level,angle,name);
    fprintf(outfile,"x%d%d = 0:1:%d;\n",level,angle,h_size-1);
    fprintf(outfile,"y%d%d = 0:1:%d;\n",level,angle,v_size-1);
    fprintf(outfile,"v%d%d = [",level,angle);
    for(j=0; j < v_size ; j++){
      for(i=0; i < h_size ; i++){
indice = j*h_size + i;
fprintf(outfile,"%.6lf ",in[indice]);
      }
      fprintf(outfile,"\n");
    }
    fprintf(outfile,"];\n");
    fprintf(outfile,"figure(1)\n");
    fprintf(outfile,"clf\n");
    fprintf(outfile,"surf(x%d%d,y%d%d,v%d%d),shading interp, view(2), colormap(jet
    level,angle);
    fprintf(outfile,"set(gca,'YDir','reverse')\n");
    fprintf(outfile,"title('Wavelets Coef Level %d Angle %d Figure %s')\n",
```

```
    level,angle,name);
    fprintf(outfile,"title('Wavelets Coef Level %d Figure %s')\n",
    level,name);
    fprintf(outfile,"xlabel('x')\n");
    fprintf(outfile,"ylabel('y')\n");
    fprintf(outfile,"zlabel('coef')\n");
    fprintf(outfile,"axis([0 63 0 63])\n");
  }
  fclose(outfile);
  return;
}

void write_message_and_pause(char *message)
{
  printf(message);
  printf("\nPress <Enter> to continue.");
  getchar();
}

void display_buffer_on_screen(MIL_ID *Display, MIL_ID *Image,
                              unsigned char *data, char *message,
                              int wait)
{
  MbufClear(*Image,0L);
  MbufPut(*Image,data);
  MdispSelect(*Display, *Image);
  if(wait)write_message_and_pause(message);
}
void copy_double_to_double(double *in, double *out)
{
  int i,j;
  for(i=0;i < IMAGE_WIDTH; i++)
    for(j=0; j < IMAGE_HEIGHT; j++)
      out[j*IMAGE_WIDTH+i] = in[j*IMAGE_WIDTH+i];
}

void copy_double_to_double_level(double *in, double *out, int h_size, int v_si
{
  int i,j;
  for(i=0; i < h_size; i++)
    for(j=0; j < v_size; j++)
      out[j*h_size+i] = in[j*h_size+i];
```

```
  return;
}
void copy_double_to_char_rev(double *in, unsigned char *out,double scale)
{
  int i,j;
  for(i=0;i < IMAGE_WIDTH; i++)
    for(j=0; j < IMAGE_HEIGHT; j++)
      out[j*IMAGE_WIDTH+i] = 0xffff - (unsigned char)
        (scale * fabs(in[j*IMAGE_WIDTH+i]) + 0.5);
}

void copy_double_to_char_rev_level(double *in, unsigned char *out,double scale,
     int h_size, int v_size)
{
  int i,j;
  for(i=0;i < h_size; i++)
    for(j=0; j < v_size; j++)
      out[j*h_size+i] = 0xffff - (unsigned char)
        (scale * fabs(in[j*h_size+i]) + 0.5);
}

void copy_double_to_char(double *in, unsigned char *out,double scale)
{
  int i,j;
  for(i=0;i < IMAGE_WIDTH; i++)
    for(j=0; j < IMAGE_HEIGHT; j++)
      out[j*IMAGE_WIDTH+i] = (unsigned char)
        (scale * fabs(in[j*IMAGE_WIDTH+i]) + 0.5);
}
void copy_double_to_char_level(double *in, unsigned char *out,double scale,
 int h_size, int v_size)
{
  int i,j;
  for(i=0;i < h_size; i++)
    for(j=0; j < v_size; j++)
      out[j*h_size+i] = (unsigned char)
        (scale * fabs(in[j*h_size+i]) + 0.5);
}

void copy_char_to_double(unsigned char *in, double *out)
{
  int i,j;
```

```
  for(i=0;i < IMAGE_WIDTH; i++)
    for(j=0; j < IMAGE_HEIGHT; j++)
      out[j*IMAGE_WIDTH+i] = (double) in[j*IMAGE_WIDTH+i];
}

int distance(double *in, int i, int j, int h_size, int v_size)
{
  if ((sqrt(SQR(h_size/2-i)+SQR(v_size/2-j))) >= (h_size/2)){
    in[j*h_size+i] == 0.0;
    return FALSE;}
  return TRUE;
}


int neighbornotzero(double *in, int i, int j, int h_level_size,
      int v_level_size)
{
  int k,l;

  for(l=(j==0 ? j : j-1); l <= (j==v_level_size-1 ? j : j+1) ; l++)
    for(k=(i==0 ? i : i-1); k <= (i==h_level_size-1 ? i : i+1) ; k++){
      if (in[l*h_level_size+k] == 0.0) return FALSE;}
  return TRUE;
}

/****QuickSort****/

void Swap(Feature_Node *elem1Ptr, Feature_Node *elem2Ptr)
{
  Feature_Node temp = *elem1Ptr;
  *elem1Ptr = *elem2Ptr;
  *elem2Ptr =  temp;
  return;
}

void Partition(Feature_Node *array, int First, int Last, int *PivotIndex)
{
  /*initially, everything but Pivot is in unknown*/
  double Pivot = SQR(array[First].point.x) + SQR(array[First].point.y);

  /*assumes Pivot is first item*/
```

```
  int LastS1 = First; /*index of last item in Si*/
  int FirstUnknown = First + 1; /*index of first item in unknown*/

  /*move one item at a time until unknown region is empty */
  for (; FirstUnknown <= Last; ++FirstUnknown)
    { /*Invariant: array[First+1..LastS1] < Pivot
      array[LastSl+1..FirstUnknown-1] >= Pivot*/

      /*move item from unknown to proper region */
      if (SQR(array[FirstUnknown].point.x) + SQR(array[FirstUnknown].point.y)
  < Pivot)
{ /* item from unknown belongs in S1*/
  ++LastS1;
  Swap(&array[FirstUnknown], &array[LastS1]);
}

      /* else item from unknown belongs in S2*/
    }   /* end for*/

    /*place Pivot in proper position and mark its location */
    Swap(&array[First], &array[LastS1]);
    *PivotIndex = LastS1;
}   /*end partition*/

/*Sorts the items in an array in
precondition: array[First..Last] is an array.
postcondition: array[First..Last] is sorted.
Calls: partition.*/

void QuickSort(Feature_Node *array, int First, int Last)
{
  int PivotIndex;

  if (First < Last)
    {  /*create the partition: S1, Pivot, S2 */
      Partition(array, First, Last, &PivotIndex);

      /*sort regions S1 and S2 */
      QuickSort(array, First, PivotIndex - 1);
      QuickSort(array, PivotIndex + 1, Last);
    }   /*end if*/
}
```

```
/****endQuickSort****/

Feature_Node *order_by_distance(Feature_Node *feature_list)
{
  Feature_Node *ordered_list;
  int k;

  ordered_list = (Feature_Node *)
    malloc(SIZE_LIST*sizeof(Feature_Node));
  if (ordered_list == NULL){
    printf("Out of memory\n");
    exit(0);
  }
  /* in case we need to save the original ordering by value,
     just change feature_list by ordered_list in the QuickSort call */
  for(k=0;k<SIZE_LIST;k++)
    ordered_list[k] = feature_list[k];

  QuickSort(feature_list, 0, SIZE_LIST-1);
  return ordered_list;
}


void find_list_highest(double *in, Feature_Node *feature_list,
       int h_size, int v_size)
{
  int i,j,k,temp_i = 1,temp_j = 1;
  double max = -1.5*MAX_VALUE, min = 1.5*MAX_VALUE;
  double   *temp;
  int l,m,indice;
  double sum=0.0, sumsqr=0.0;

  temp = (double *)
    malloc(h_size*v_size*IMAGE_BAND*sizeof(double));

  if (temp == NULL){
    printf("Out of memory\n");
    exit(0);
  }

  copy_double_to_double_level(in, temp, h_size, v_size);
```

```
  for(k=0;k<SIZE_LIST;k++){
    for(j=0; j < v_size ; j++)
      for(i=0; i < h_size ; i++){
indice = j*h_size + i;
if(fabs(temp[indice]) >= max
   && neighbornotzero(temp,i,j,h_size,v_size)){
  max = fabs(temp[indice]);
  temp_i = i;
  temp_j = j;
}
if(temp[indice] < min){
  min = fabs(temp[indice]);
}
      }
    feature_list[k].point.x = temp_i;
    feature_list[k].point.y = temp_j;
    feature_list[k].value = temp[temp_j*h_size + temp_i];

    /* sum and sum square, plus zeroing the point and neighbors*/
    for(m=(temp_j==0 ? temp_j : temp_j-1);
    feature_list[k].mean = sum/9;
    feature_list[k].variance = (sumsqr - SQR(feature_list[k].mean)*9)/8;

    max = min;
    sum = sumsqr = 0.0;
  }
  order_by_distance(feature_list);
  free(temp);
  return;
}

void find_list_lowest(double *in, Feature_Node *feature_list,
       int h_size, int v_size)
{
  int i,j,k,temp_i = 1,temp_j = 1;
  double max = in[0], min = in[0];
  double   *temp;
  int l,m,indice;
  double sum=0.0, sumsqr=0.0;

  temp = (double *)
```

```
   malloc(h_size*v_size*IMAGE_BAND*sizeof(double));

  if (temp == NULL){
    printf("Out of memory\n");
    exit(0);
  }

  copy_double_to_double_level(in, temp,  h_size, v_size);

  /*borders are being taken off*/
  for(k=0;k<SIZE_LIST;k++){
    for(j=1; j < v_size-1 ; j++)
      for(i=1; i < h_size-1 ; i++){
indice = j*h_size + i;
if(temp[indice] <= min && neighbornotzero(temp,i,j,h_size,v_size)){
  min = temp[indice];
  temp_i = i;
  temp_j = j;
}
if(temp[indice] > max){
  max = temp[indice];
}
      }
    feature_list[k].point.x = temp_i;
    feature_list[k].point.y = temp_j;
    feature_list[k].value = min;

    /* sum and sum square, plus zeroing the point and neighbors*/
    for(m=temp_j-1; m < temp_j+2 ; m++)
      for(l=temp_i-1; l < temp_i+2 ; l++){
indice = m*h_size+l;
sum    += temp[indice];
sumsqr += SQR(temp[indice]);
temp[indice] = 0.0;}

    feature_list[k].mean = sum/9;
    feature_list[k].variance = (sumsqr - SQR(feature_list[k].mean)*9)/8;

    min = max;
    sum = sumsqr = 0.0;
  }
  free(temp);
```

```
  return;
}


void log_of(double *inout)
{
  int i,j;
  for(i=0;i < IMAGE_WIDTH; i++)
    for(j=0; j < IMAGE_HEIGHT; j++) {
      inout[j*IMAGE_WIDTH+i] = (inout[j*IMAGE_WIDTH+i] == 0.0) ?
0.00000000000000000001 : inout[j*IMAGE_WIDTH+i];
      inout[j*IMAGE_WIDTH+i] = (double) log10(inout[j*IMAGE_WIDTH+i]);
    }
}

Point calc_centroid(Feature_Node *feature_list, int level)
{
  int i;
  double sum_x = 0.0, sum_y = 0.0;
  int max_index = SIZE_LIST;
  Point centroid;

  for(i=0; i < max_index; i++)
    {
      sum_x += feature_list[i].point.x;
      sum_y += feature_list[i].point.y;
    }
  centroid.x = sum_x/max_index;
  centroid.y = sum_y/max_index;

  return centroid;
}

int load_names(char *file_list,char **names)
{
  FILE *infile;
  int i = 0, valid;
  char *file = (char *) malloc(STRING*sizeof(char));

  infile = fopen(file_list,"r");
  if(infile == NULL){
    printf("\nError reading from file %s\n",file_list);
```

```
    exit(0);
  }else{
    while (!feof(infile)) {
      valid = fscanf(infile,"%s\n", file);
      i++;
     }
    rewind(infile);
    *names =  (char *) malloc(STRING*sizeof(char)*i);
    i = 0;
    while (!feof(infile)) {
      valid = fscanf(infile,"%s\n", (*names+i*STRING));
      i++;
    }
  }
  fclose(infile);
  free(file);
  return(i);
}

int load_names_and_types(char *file_list,Training_List **names_list)
{
  FILE *infile;
  int i = 0, valid;
  char *file = (char *) malloc(STRING*sizeof(char));
  Training_List *temp_ptr, *first;

  infile = fopen(file_list,"r");
  if(infile == NULL){
    printf("\nError reading from file %s\n",file_list);
    exit(0);
  }else{
    *names_list = NULL;
    while (!feof(infile)) {

      if(((temp_ptr = (Training_List *)malloc(sizeof(Training_List))) == NULL)
 || ((temp_ptr->file_name = (char *)malloc(STRING*sizeof(char)))
     == NULL)) {
printf("Not enough space for loading names\n");
exit(0);
      }
      valid = fscanf(infile,"%s %d\n", temp_ptr->file_name,
     &(temp_ptr->same_type));
```

```
      printf("%s %d\n", temp_ptr->file_name, temp_ptr->same_type);
      if (*names_list == NULL)
{
  first = *names_list = temp_ptr;
  temp_ptr->next = NULL;
}
      else
{
  *names_list = (*names_list)->next = temp_ptr;
  temp_ptr->next = NULL;
}
      if(temp_ptr->same_type == FALSE)
i++;
    }
    *names_list = first;
  }
  fclose(infile);
  return(i);
}

void normalize(double *inout, double factor, int h_size, int v_size)
{
  int i,j;
  for(i=0; i < h_size; i++)
    for(j=0; j < v_size; j++)
      if(factor != 0.0)
inout[j*h_size+i] = inout[j*h_size+i]/factor;
  return;
}

void init_write_lisp_training(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"w");

  if(outfile == NULL){
```

```
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,";;Training data level %d\n", level);
    fprintf(outfile,"(defun train () (main '(");
  }
  fclose(outfile);
  return;
}


void write_lisp_training(Feature_Node *feature_list, char *name,
 int level, int angle, char *file)
{
  FILE *outfile;
  int i = 0, max_index = SIZE_LIST;
  Point centroid = calc_centroid(feature_list, level);
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,"\n;;Highest_%d, level %d, angle %d, figure %s\n(",
    max_index,level,angle,name);
    for(i=0; i < max_index; i++)
      {
fprintf(outfile,"%.16lf %.16lf %.16lf %.16lf %.16lf ",
feature_list[i].point.x-centroid.x,
feature_list[i].point.y-centroid.y,feature_list[i].value,
feature_list[i].mean,feature_list[i].variance);
      }
    fprintf(outfile,")\n");
  }
```

```
  fclose(outfile);
  return;
}

void print_bit(int value, FILE *outfile, int nn)
{
  int i;
  int mask = 1 << (nn-1);

  for(i=1; i<=nn; i++)
    {
      fputc(((value & mask) == 0) ? '0' :'1', outfile);
      value <<= 1;
      fputc(' ',outfile);
    }
  return;
}

int Int(double value)
{
  int temp = value;

  if (value == (double) temp)
    return temp;
  else
    return temp+1;
}

void end_write_lisp_training(char *file, int num_files, Training_List *names,
     int level)
{
  int i = 0, angle;
  double ii = 1./2;
  FILE *outfile;
  int num_bit1 = Int(log10(num_files)/log10(2));
  int num_bit2 = Int(log10(360/STEP_ANGLE)/log10(2));
  char file_n1[STRING], file_n2[STRING];

  num_bit1 = num_files;
  strcpy(file_n1,file);
  sprintf(file_n2,"1%d",level);
  strcat(file_n1,file_n2);
```

```
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,")\n;;outputs \n'(");
    do{
      /*if the file has the same output as the previous, don't increment i*/
      if(names->same_type == FALSE){
ii *= 2;
i = (int) ii;
      }
      for(angle=0; angle < 360/STEP_ANGLE; angle++){
fprintf(outfile,"(");
print_bit(i,outfile,num_bit1);
print_bit(angle,outfile,num_bit2);
fprintf(outfile,")\n");
      }
    }while((names = names->next) != NULL);
    fprintf(outfile,")))\n");
  }
  fclose(outfile);
  return;
}

void write_lisp_id_file(char *file, int num_files, Training_List *names,
int level)
{
  int i = 1, angle;
  FILE *outfile;
  int num_bit1 = Int(log10(num_files)/log10(2));
  int num_bit2 = Int(log10(360/STEP_ANGLE)/log10(2));
  char file_n1[STRING], file_n2[STRING];
  char *name;

  num_bit1 = num_files;
  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
```

```
  strcat(file_n1,".lisp");

  sprintf(file_n2,"");

  outfile = fopen(file_n1,"w");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,";;identification list for level %d \n", level);
    fprintf(outfile,"(setf *id_list%d* '(", level);

    do{
      /*if the file has the same output as the previous, skip it*/
      if(names->same_type == FALSE){
name = names->file_name;
for(angle=0; angle < 360/STEP_ANGLE; angle++){
  strcpy(file_n1,name);
  sprintf(file_n2,"l%da%d",level,angle*STEP_ANGLE);
  strcat(file_n1,file_n2);
  fprintf(outfile,"(%s ",file_n1);
  fprintf(outfile,"(");
  print_bit(i,outfile,num_bit1);
  print_bit(angle,outfile,num_bit2);
  fprintf(outfile,"))\n");
}
i *= 2;
      }
    }while((names = names->next) != NULL);
    fprintf(outfile,"))\n");
  }
  fclose(outfile);
  return;
}

void free_file_list(Training_List **names_list)
{
  Training_List *temp_ptr;
  while((*names_list) != NULL){
    free((*names_list)->file_name);
```

```
    temp_ptr = *names_list;
    *names_list = (*names_list)->next;
    free(temp_ptr);
  }
  return;
}


/*********************************************************************/
/*              Phase 2 - Testing                                    */
/*********************************************************************/

void init_write_lisp_test(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"w");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,";;Test data level %d\n", level);
    fprintf(outfile,"(defun test () (recon '(");
  }
  fclose(outfile);
  return;
}

void write_lisp_test(Feature_Node *feature_list, char *name,
 int level, char *file)
{
  FILE *outfile;
  int i = 0, max_index = SIZE_LIST;
  Point centroid = calc_centroid(feature_list, level);
  char file_n1[STRING], file_n2[STRING];
```

```
  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,"\n;;Highest_%d, level %d, figure %s\n(",
    max_index,level,name);
    for(i=0; i < max_index; i++)
      {
fprintf(outfile,"%.16lf %.16lf %.16lf %.16lf %.16lf ",
feature_list[i].point.x-centroid.x,
feature_list[i].point.y-centroid.y,feature_list[i].value,
feature_list[i].mean,feature_list[i].variance);
      }
    fprintf(outfile,")\n");
  }
  fclose(outfile);
  return;
}

void end_write_lisp_test(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
```

```
  else{
    fprintf(outfile,")))\n");
  }
  fclose(outfile);
  return;
}


/*********************************************************************/
/*              Phase 3 - Scanning                                   */
/*********************************************************************/


int neighbornotzero_scan(double *in, int i, int j, int level)
{
  int k,l;
  int h_level_size = IMAGE_WIDTH/(pow(2,level));
  int v_level_size = IMAGE_HEIGHT/(pow(2,level));

  for(l=(j==0 ? j : j-1); l <= (j==v_level_size-1 ? j : j+1) ; l++)
    for(k=(i==0 ? i : i-1); k <= (i==h_level_size-1 ? i : i+1) ; k++){
      if (in[l*h_level_size+k] == 0.0) return FALSE;}
  return TRUE;
}


void histogram(double *in, int level, double level_max)
{
  int i,j,indice,threshold,ten_percent,count = 0;
  int *hist;
  int h_level_size = IMAGE_WIDTH/(pow(2,level));
  int v_level_size = IMAGE_HEIGHT/(pow(2,level));

  hist = (int *)
    malloc((int) (level_max*sizeof(int)));
  ten_percent = (int) (0.1*h_level_size*v_level_size);

  printf("(int) fabs(level_max) = %d\n",(int) fabs(level_max));

  for (i=0;i<(int)level_max;i++)
    hist[i] = 0;
  for(j=0; j < v_level_size ; j++)
```

```
    for(i=0; i < h_level_size ; i++)
      {
indice = (int) fabs(in[j*h_level_size + i]);
hist[indice]++;
      }
  i = -1;
  printf("ten_percent = %d\n",ten_percent);
  while(count < ten_percent)
    {
      i++;
      count += hist[i];
      printf("hist[%d] = %d\n",i,hist[i]);
    }
  threshold = i;
  printf("threshold = %d\n",threshold);

  for(j=0; j < v_level_size ; j++)
    for(i=0; i < h_level_size ; i++)
      {
indice = j*h_level_size + i;
if ((int)fabs(in[indice]) <= threshold)
  {
    in[indice] = 0.0;
  }
      }
  return;
}

void find_scan_list_highest(double *in, Feature_Node *feature_list,
    int h_init, int h_size, int v_init, int v_size,
    int level)
{
  int i,j,k,temp_i = 1,temp_j = 1;
  double max = -1.5*MAX_VALUE, min = 1.5*MAX_VALUE;
  double win_max = -1.5*MAX_VALUE, win_min = 1.5*MAX_VALUE;
  double   *temp;
  int l,m,indice;
  double sum=0.0, sumsqr=0.0;
  int h_level_size = IMAGE_WIDTH/(pow(2,level));
  int v_level_size = IMAGE_HEIGHT/(pow(2,level));

  temp = (double *)
```

```
    malloc(h_level_size*v_level_size*IMAGE_BAND*sizeof(double));

  if (temp == NULL){
    printf("Out of memory\n");
    exit(0);
  }

  copy_double_to_double_level(in, temp, h_level_size, v_level_size);

  for(j=v_init; j < v_init+v_size; j++)
    for(i=h_init; i < h_init+h_size; i++){
      indice = j*h_level_size + i;
      if(fabs(temp[indice]) > win_max)
win_max = fabs(temp[indice]);
      if(fabs(temp[indice]) < win_min)
win_min = fabs(temp[indice]);
    }

  normalize(temp, win_max/MAX_VALUE, h_level_size, v_level_size);

  /*borders are being taken off*/
  for(k=0;k<SIZE_LIST;k++){
    for(j=v_init; j < v_init+v_size; j++)
      for(i=h_init; i < h_init+h_size; i++){
indice = j*h_level_size + i;
if(fabs(temp[indice]) >= max &&
   neighbornotzero_scan(temp,i,j,level)){
  max = fabs(temp[indice]);
  temp_i = i;
  temp_j = j;
}
if(temp[indice] < min){
  min = fabs(temp[indice]);
}
      }
    feature_list[k].point.x = temp_i;
    feature_list[k].point.y = temp_j;
    feature_list[k].value = temp[temp_j*h_level_size + temp_i];

    /* sum and sum square, plus zeroing the point and neighbors*/
    for(m=(temp_j==0 ? temp_j : temp_j-1);
m <= (temp_j==v_level_size-1 ? temp_j : temp_j+1) ; m++)
```

```
      for(l=(temp_i==0 ? temp_i : temp_i-1);
  l <= (temp_i==h_level_size-1 ? temp_i : temp_i+1) ; l++){
indice = m*h_level_size+l;
/*average is being calculated with the abs values */
sum    += fabs(temp[indice]);
sumsqr += SQR(temp[indice]);
temp[indice] = 0.0;
      }
    feature_list[k].mean = sum/9;
    feature_list[k].variance = (sumsqr - SQR(feature_list[k].mean)*9)/8;

    max = min;
    sum = sumsqr = 0.0;
  }
  order_by_distance(feature_list);
  for(k=0;k<SIZE_LIST;k++)
    {
      if(feature_list[k].value == 0.0)
/* flag that signals the NN to discard this template */
feature_list[0].value = 0.0;
    }
  free(temp);
  return;
}

/* Output for Matlab Plotting*/
void write_wav_scan_level_high_to_disk(Feature_Node *feature_list, char *name,
     int level, int h_size, int v_size, int window,
       int h_init, int v_init)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];
  int i, temp_x, temp_y, max_index = SIZE_LIST;
  Point centroid = calc_centroid(feature_list, level);
  int h_level_size = IMAGE_WIDTH/(pow(2,level));
  int v_level_size = IMAGE_HEIGHT/(pow(2,level));

  strcpy(file_n1,name);
  sprintf(file_n2,"l%dhi%dvi%dhs%dvs%d",level,h_init,v_init,h_size,v_size);
  strcat(file_n1,file_n2);
  strcat(file_n1,".m");
  outfile = fopen(file_n1,"w");
```

```
  if(outfile == (FILE *)-1){
    printf("\nError writing wavelets to files %s\n",file_n1);
    exit(0);
  }
  else{
    fprintf(outfile,"%%Highest_%d, level %d, figure %s\n",
    max_index,level,name);
    fprintf(outfile,"xh%d = 0:1:%d;\n",level,h_level_size-1);
    fprintf(outfile,"yh%d = 0:1:%d;\n",level,v_level_size-1);
    fprintf(outfile,"value%d = zeros(%d,%d);\n",level,v_level_size,h_level_size)
    fprintf(outfile,"mean%d = zeros(%d,%d);\n",level,v_level_size,h_level_size);
    fprintf(outfile,"variance%d = zeros(%d,%d);\n",level,v_level_size,h_level_si
    for(i=0; i < max_index; i++)
      {
temp_x = feature_list[i].point.x + 1;
temp_y = feature_list[i].point.y + 1;
fprintf(outfile,"value%d(%d,%d) = %.6lf;\n",level,temp_y,temp_x,
feature_list[i].value);
fprintf(outfile,"mean%d(%d,%d) = %.6lf;\n",level,temp_y,temp_x,
feature_list[i].mean);
fprintf(outfile,"variance%d(%d,%d) = %.6lf;\n",level,temp_y,temp_x,
feature_list[i].variance);
      }
    fprintf(outfile, "xh%d = xh%d - %.6lf;\n",level,
    level,centroid.x);
    fprintf(outfile, "yh%d = yh%d - %.6lf;\n",level,
    level,centroid.y);
    fprintf(outfile,"figure(2)\n");
    fprintf(outfile,"clf\n");
    fprintf(outfile,"surf(xh%d,yh%d,value%d),shading interp, view(2), colormap(j
    fprintf(outfile,"set(gca,'YDir','reverse')\n");
    fprintf(outfile,"title('Highest_%d, level %d, figure %s')\n",
    max_index,level,name);
    fprintf(outfile,"xlabel('x')\n");
    fprintf(outfile,"ylabel('y')\n");
    fprintf(outfile,"zlabel('coef')\n");
    fprintf(outfile,"figure(3)\n");
    fprintf(outfile,"clf\n");
    fprintf(outfile,"surf(xh%d,yh%d,abs(value%d)),shading interp, view(2), color
    fprintf(outfile,"set(gca,'YDir','reverse')\n");
    fprintf(outfile,"title('Highest_%d, level %d, figure %s')\n",
    max_index,level,name);
```

```
    fprintf(outfile,"xlabel('x')\n");
    fprintf(outfile,"ylabel('y')\n");
    fprintf(outfile,"zlabel('abs(coef)')\n");
  }
  fclose(outfile);
  return;
}
void init_write_lisp_scan(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"1%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"w");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,";;Scan data level %d\n", level);
    fprintf(outfile,"(defun scan () (recon '(");
  }
  fclose(outfile);
  return;
}

void write_lisp_scan(Feature_Node *feature_list, char *name, int level,
     int h_init, int h_size, int v_init, int v_size)
{
  FILE *outfile;
  int i = 0, max_index = SIZE_LIST;
  Point centroid = calc_centroid(feature_list, level);
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,name);
  sprintf(file_n2,"1%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
```

```
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file_n1);
    exit(0);
  }
  else{
    fprintf(outfile,"\n;;I_col %d, I_row %d, h_size %d, v_size %d\n(",
    h_init,v_init,h_size,v_size);
    for(i=0; i < max_index; i++)
      {
fprintf(outfile,"%.16lf %.16lf %.16lf %.16lf %.16lf ",
feature_list[i].point.x-centroid.x,
feature_list[i].point.y-centroid.y,feature_list[i].value,
feature_list[i].mean,feature_list[i].variance);
      }
    fprintf(outfile,")\n");
  }
  fclose(outfile);
  return;
}

void end_write_lisp_scan(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,")))\n");
  }
  fclose(outfile);
  return;
```

```
}


void init_write_lisp_win_id(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"w");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,";;windows list for level %d \n", level);
    fprintf(outfile,"(setf *win_id%d* '(\n", level);
  }
  fclose(outfile);
  return;
}

void write_lisp_win_id(char *file, int level,
       int h_init, int h_size, int v_init, int v_size)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
```

```
  else{
    fprintf(outfile,"(level %d i_col %d i_row %d h_size %d v_size %d)\n",
    level,h_init,v_init,h_size,v_size);
  }
  fclose(outfile);
  return;
}


void end_write_lisp_win_id(char *file, int level)
{
  FILE *outfile;
  char file_n1[STRING], file_n2[STRING];

  strcpy(file_n1,file);
  sprintf(file_n2,"l%d",level);
  strcat(file_n1,file_n2);
  strcat(file_n1,".lisp");
  outfile = fopen(file_n1,"a");

  if(outfile == NULL){
    printf("\nError writing to file %s\n",file);
    exit(0);
  }
  else{
    fprintf(outfile,"))\n");
  }
  fclose(outfile);
  return;
}

void scan_picture(double *in, Feature_Node *out, int level, int h_level_size,
  int v_level_size, char *name, char *win_id)
{
  int window = 0;
  int h_init = 0, h_size = h_level_size, v_init = 0, v_size = v_level_size;
  int win_min = h_level_size >> SCAN_WIN;

  init_write_lisp_scan(name, level);
  init_write_lisp_win_id(win_id, level);

  /*rectangular template for b45*/
```

```
  v_size = 28;
  h_size = 36;
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }

  /*rectangular template for f45*/
  v_size = 36;
  h_size = 45;
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }

   /*rectangular template for k45*/
  v_size = 27;
  h_size = 34;
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
```

```
    }


  /*rectangular vertical scanning*/
  v_size = v_level_size >> 2;
  h_size = h_level_size;
  for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP)
    for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }
  /*end rectangular vertical scanning*/

  /*rectangular horizontal scanning*/
  v_size = v_level_size;
  h_size = h_level_size >> 2;
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }
  /*end rectangular horizontal scanning*/

  /*rectangular vertical scanning*/
  h_size = (int)(3.0/4.0*h_level_size);
  v_size = v_level_size >> 2;
  for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP)
    for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
```

```
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }
  /*end rectangular vertical scanning*/

  /*rectangular horizontal scanning*/
  h_size = h_level_size >> 2;
  v_size = (int)(3.0/4.0*v_level_size);
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }
  /*end rectangular horizontal scanning*/

  h_size = (int)(h_level_size*0.6);
  v_size = v_level_size >> 2;
  for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP)
    for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP){

      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }
  h_size = h_level_size >> 2;
  v_size = (int)(v_level_size*0.6);
  for(v_init = 0; v_init+v_size <= v_level_size; v_init+=SCAN_STEP)
    for(h_init = 0; h_init+h_size <= h_level_size; h_init+=SCAN_STEP){
```

```
      find_scan_list_highest(in, out, h_init, h_size, v_init, v_size,
     level);
      write_wav_scan_level_high_to_disk(out, name, level, h_size, v_size,
   window, h_init, v_init);
      write_lisp_scan(out, name, level, h_init, h_size, v_init, v_size);
      write_lisp_win_id(win_id, level, h_init, h_size, v_init, v_size);
      window++;
    }

   end_write_lisp_scan(name, level);
   end_write_lisp_win_id(win_id, level);
  return;
}
```

APPENDIX E. LISP CODE FOR RECOGNITION

```
;*****************************************************************
; File: recon.lisp
; Name:Jader Gomes da Silva Filho
; Date:08/14/97
; Operating Environment: SUN
; Language: LISP
;*****************************************************************


(defun initialize (weights ids win_ids)
  (load_weights weights)
  (load_ids ids)
  (load_win_id win_ids)
)

;;-------------------------------------------------
;; Load values for the weights
;;

(defun load_weights (weights)
  (setf pathname (make-pathname :name weights))
  (with-open-file (str pathname :direction :input :if-does-not-exist :error)
  (do ((expression (read str nil 'eof)
   (read str nil 'eof)))
      ((eql expression 'eof))
    (eval expression)))
)

;;-------------------------------------------------
;; Load values for the inputs
;;

(defun load_inputs (inputs)
  (setf pathname (make-pathname :name inputs))
  (with-open-file (str pathname :direction :input :if-does-not-exist :error)
  (do ((expression (read str nil 'eof)
   (read str nil 'eof)))
      ((eql expression 'eof))
    (eval expression)))
)

;;-------------------------------------------------
```

```
;; Load values for the inputs
;;

(defun load_ids (ids)
  (setf pathname (make-pathname :name ids))
  (with-open-file (str pathname :direction :input :if-does-not-exist :error)
  (do ((expression (read str nil 'eof)
   (read str nil 'eof)))
      ((eql expression 'eof))
    (eval expression)))
)

;;-------------------------------------------------
;; Load values for the inputs
;;

(defun load_win_id (win_ids)
  (setf pathname (make-pathname :name win_ids))
  (with-open-file (str pathname :direction :input :if-does-not-exist :error)
  (do ((expression (read str nil 'eof)
   (read str nil 'eof)))
      ((eql expression 'eof))
    (eval expression)))
)



;;-------------------------------------------------
;;Genetates a list of repeated inputs of the same size of
;;the weigth list.
;;
(defun make_input_list (input w_list)
  (list_of (length w_list) input))

;;-------------------------------------------------
;; this provides a standard activation
;; function for a neural net--the logistic
;; sigmoid function
;; f(x) = 1 / (1 + e (-x))
;;
;;
(defun sigmoid (x)
   (/ ( + 1 (exp (- x)))))
```

```
;;-------------------------------------------------
;; this does summation on a vector
;;
;; > (summation vector)
;; > (summation '(2.0 3.0 2.6))
;; 7.6
;;
(defun summation (vector)
   (eval
        (cons '+ vector)))

;;-------------------------------------------------
;; this does multiplication of two vectors
;;
;; > (vector_multiply '(0 1 1) '(-4.8 5.2 -2.3))
;; (0 5.2 -2.3)
;;
(defun vector_multiply (vector1 vector2)
     (mapcar #'* vector1 vector2))

;;-------------------------------------------------
;; this makes a list n elements long of elt
;;
;; > (list-of 3 2.0)
;; (2.0 2.0 2.0)
;;
(defun list_of (n elt)
   (if (zerop n)
        nil
       (cons elt (list_of ( - n 1) elt))))

;;-------------------------------------------------
;; this does sum of two vectors
;;
;; > (vector_sum '(0 1 1) '(-4.8 5.2 -2.3))
;; (-4.8 6.2 -1.3)
;;
(defun vector_sum (vector1 vector2)
     (mapcar #'+ vector1 vector2))

;;-------------------------------------------------
```

```
;;Evaluates one node according to a list of inputs
;;and a list of weigths for that node
;; > (node_eval '(0 0 1) '(-4.8  4.6 -2.6))
;; 0.06913843

(defun node_eval (inputs weigths)
  (sigmoid (summation (vector_multiply inputs weigths))))

;;-------------------------------------------------
;;Evaluates one level, generating a list of values
;;that will be the input for the next level
;;
;; > (level_eval '(0 0) inp_wlist)
;; (0.06913843 0.03916572)

(defun level_eval (input w_list)
  (mapcar #'node_eval (make_input_list input w_list) w_list))


;;-------------------------------------------------
;; Does the usual forward evaluation, saving the values
;; of the hidden level
(defun forward_eval (input w_list1 w_list2)
  (if (= (nth 2 input) 0.0)
      nil
    (level_eval (level_eval input w_list1) w_list2))
  )


;;-------------------------------------------------
;; output results
;;
(defun print_results (input output nice_output)
  (format t "~%Input:~A~%Output:~A~%Nice Output:~A~%" input output nice_output)
  (setf pathname (make-pathname :name "output.dat"))
  (setf outfile (open pathname :direction :output :if-exists :rename
      :if-does-not-exist :create))
  (format outfile "~%Input:~A~%Output:~A~%Nice Output:~A~%"
  input output nice_output)
  (close outfile)
)
```

```
;;-------------------------------------------------
;; remove-element
;;(remove-element 0 '(1 2 3 4))
;;(2 3 4)
(defun remove-element (elem list)
  (append (subseq list 0 elem) (subseq list (1+ elem))))



;;-------------------------------------------------
;; Does the usual forward evaluation
;;
;;(forward_eval_inputs '((2 3 0) (2 4 5)) '((.5 .4 .3)(.5 .4 .3)(.5 .4 .3))
;;                                        '((.5 .1 .2)(.5 .1 .2)))
;;((0.6768457 0.6768457) (0.6871778 0.6871778))

(defun forward_eval_inputs (input w_list1 w_list2)
  (if (car input)
      (append (list (forward_eval (car input) w_list1 w_list2))
      (forward_eval_inputs (rest input) w_list1 w_list2))
    nil
    )
  )

;;-------------------------------------------------
;; User Round function
;;
(defun user_round (input)
  (let ((value (car input)))
    (if value
(if (< value 0.5)
    (cons (floor value) (user_round (rest input)))
  (cons (ceiling value) (user_round (rest input))))
      nil)
    )
)


;;-------------------------------------------------
;; Round the results to 0's and 1's
;;
;;(round_outputs '((0.6871778 0.6871778) nil (0.6871778 0.6871778)))
```

```
;;((1 1) nil (1 1))

(defun round_outputs (output)
  (mapcar #'user_round output)
)

;;-------------------------------------------------
;; CALCULATE ERROR VECTOR FOR OUTPUT NODES
;; this compares calculated output with
;; expected output. And save the errors in the
;; variable SAVED_ERROR_OUT
;;
;; (calc_output_error '(1.0 0.0) '(0.88 0.13))
;; 0.08845903
(defun calc_output_error (exp_output calc_output)
  (setf SAVED_ERROR_OUT
(mean_sq_error (mapcar #'- exp_output calc_output))))



;;-------------------------------------------------
;; function user to calculate the square root of the
;; mean of the errors
(defun mean_sq_error (w_list)
  (if (null w_list)
      nil
    (/ (sqrt(summation(vector_multiply w_list w_list))) (length w_list))))

;;-------------------------------------------------
;; function user to calculate the mean error of several
;; inputs
;; (error_list '((1.0 0.0)) '((0.88 0.13)))
;; (0.08845903)
(defun error_list (output nice_output)
  (mapcar #'calc_output_error output nice_output)
)

(defun find_first (inp_list)
  (find-if #'(lambda (x)
       (< x 0.003))
   inp_list)
)
```

```
;;-------------------------------------------------
;; Find best id
(defun find_best (id_list)
  (cond
   ((null id_list)
    (cond
  ((and (car sol8) (< (car sol8) solthres)) (cons 8 sol8))
  ((and (car sol7) (< (car sol7) solthres)) (cons 7 sol7))
  ((and (car sol6) (< (car sol6) solthres)) (cons 6 sol6))
  ((and (car sol5) (< (car sol5) solthres)) (cons 5 sol5))
  ((and (car sol4) (< (car sol4) solthres)) (cons 4 sol4))
  ((and (car sol3) (< (car sol3) solthres)) (cons 3 sol3))
  ((and (car sol2) (< (car sol2) solthres)) (cons 2 sol2))
  ((and (car sol1) (< (car sol1) solthres)) (cons 1 sol1))))

   ((and
     (caadr id_list) (caaddr id_list) (caar (cdddr id_list))
     (caar (cddddr id_list)) (caar (nthcdr 5 id_list))
     (caar (nthcdr 6 id_list)) (caar (nthcdr 7 id_list))
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list)))
     (< (abs (- (caar id_list) (caar (cdddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadddr id_list)))
     (< (abs (- (caar id_list) (caar (cddddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (cddddr id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 5 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 5 id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 6 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 6 id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 7 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 7 id_list)))))
    (if (and (< (caar id_list) better8) (> (caar id_list) thres))
(setf better8 (caar id_list)
      sol8 (nth 4 id_list)))
    (find_best (nthcdr 8 id_list)))

   ((and
     (caadr id_list) (caaddr id_list) (caar (cdddr id_list))
     (caar (cddddr id_list)) (caar (nthcdr 5 id_list))
```

```
     (caar (nthcdr 6 id_list))
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list)))
     (< (abs (- (caar id_list) (caar (cdddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadddr id_list)))
     (< (abs (- (caar id_list) (caar (cddddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (cddddr id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 5 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 5 id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 6 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 6 id_list)))))
    (if (and (< (caar id_list) better7) (> (caar id_list) thres))
(setf better7 (caar id_list)
      sol7 (nth 3 id_list)))
    (find_best (nthcdr 7 id_list)))
   ((and
     (caadr id_list) (caaddr id_list) (caar (cdddr id_list))
     (caar (cddddr id_list)) (caar (nthcdr 5 id_list))
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list)))
     (< (abs (- (caar id_list) (caar (cdddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadddr id_list)))
     (< (abs (- (caar id_list) (caar (cddddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (cddddr id_list))))
     (< (abs (- (caar id_list) (caar (nthcdr 5 id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (nthcdr 5 id_list)))))
    (if (and (< (caar id_list) better6) (> (caar id_list) thres))
(setf better6 (caar id_list)
      sol6 (nth 3 id_list)))
    (find_best (nthcdr 6 id_list)))
   ((and
     (caadr id_list) (caaddr id_list) (caar (cdddr id_list))
     (caar (cddddr id_list))
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list)))
     (< (abs (- (caar id_list) (caar (cdddr id_list)))) acceptdif)
```

```
     (equal (nth 2 (car id_list)) (nth 2 (cadddr id_list)))
     (< (abs (- (caar id_list) (caar (cddddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (car (cddddr id_list)))))
    (if (and (< (caar id_list) better5) (> (caar id_list) thres))
(setf better5 (caar id_list)
      sol5 (nth 2 id_list)))
    (find_best (nthcdr 5 id_list)))
   ((and
     (caadr id_list) (caaddr id_list) (caar (cdddr id_list))
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list)))
     (< (abs (- (caar id_list) (caar (cdddr id_list)))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadddr id_list))))
    (if (and (< (caar id_list) better4) (> (caar id_list) thres))
(setf better4 (caar id_list)
      sol4 (nth 2 id_list)))
    (find_best (nthcdr 4 id_list)))
   ((and
     (caadr id_list) (caaddr id_list)
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list)))
     (< (abs (- (caar id_list) (caaddr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (caddr id_list))))
    (if (and (< (caar id_list) better3) (> (caar id_list) thres))
(setf better3 (caar id_list)
      sol3 (nth 1 id_list)))
    (find_best (nthcdr 3 id_list)))
   ((and
     (caadr id_list)
     (< (abs (- (caar id_list) (caadr id_list))) acceptdif)
     (equal (nth 2 (car id_list)) (nth 2 (cadr id_list))))
    (if (and (< (caar id_list) better2) (> (caar id_list) thres))
(setf better2 (caar id_list)
      sol2 (nth 1 id_list)))
    (find_best (nthcdr 2 id_list)))
   ((and (< (caar id_list) better1) (> (caar id_list) thres))
    (setf better1 (caar id_list)
  sol1 (nth 0 id_list))
    (find_best (cdr id_list)))
   (t (find_best (cdr id_list)))
```

```
   )
  )


;;-------------------------------------------------
;; Find best id
(defun find_best_id (output nice_output)
  (let*
      ((list_error (error_list output nice_output))
       (minima (find_first list_error))
        (pos (position minima list_error :test #'equal))
       (result (nth pos nice_output)))
    (if (< minima 0.5)
(list 'Error minima (nth pos *win_id3*)
      (car (nth
    (position result *id_list3* :key #'cadr :test #'equal)
    *id_list3*)))
      '(NO_RESULTS 1)))
  )

;;-------------------------------------------------
;; Find id
(defun find_id (output nice_output)
  (mapcar #'(lambda (e x y)
      (if e
  (list e y (car
     (nth
      (position x *id_list3* :key #'cadr
:test #'equal) *id_list3*)))
nil))
  (error_list output nice_output) nice_output *win_id3*)
  )

;;(find_id '((0.6871778 0.6871778) nil (0.6871778 0.6871778))
;;'((1 1) nil (1 1)))
;;(error_list '((0.6871778 0.6871778) nil (0.6871778 0.6871778))
;;'((1 1) nil (1 1)))
;;(0.22119872 nil 0.22119872)

;; Utility Functions to count the number of identifications of a kind
(defun compress (x)
  (if (consp x)
```

```
      (compr (car x) 1 (cdr x) (caar x))
    x))

(defun compr (elem n lst summ)
  (if (null lst)
      (list (n-elems elem n summ))
    (let ((next (car lst)))
      (if (string= (nth 2 next) (nth 2 elem))
  (compr elem (+ n 1) (cdr lst) (+ (nth 0 next) summ))
(cons (n-elems elem n summ)
      (compr next 1 (cdr lst) (car next)))))))

(defun n-elems (elem n summ)
  (list (/ summ n) n elem)
  )



;;-------------------------------------------------
;; Function to find the best result for a single NN System
;;

(defun recon (input)
  (setf thres1 0.0)
  (setf thres 0.0)
  (setf solthres 0.5)
  (setf acceptdif 1e-8)
  (setf better1 1.0)
  (setf sol1 '())
  (setf better2 1.0)
  (setf sol2 '())
  (setf better3 1.0)
  (setf sol3 '())
  (setf better4 1.0)
  (setf sol4 '())
  (setf better5 1.0)
  (setf sol5 '())
  (setf better6 1.0)
  (setf sol6 '())
  (setf better7 1.0)
  (setf sol7 '())
  (setf better8 1.0)
```

```
  (setf sol8 '())
  (setf best_result '())
  (setf output (forward_eval_inputs input inp_wlist hid_wlist))
  (setf nice_output (round_outputs output))
  (setf id_list_ori (find_id output nice_output))

  (setf bestname (make-pathname :name "idsori.txt"))
  (with-open-file (str bestname :direction :output :if-exists :rename
       :if-does-not-exist :create)
  (format str "Ids = ~A~%~%" id_list_ori))

;; this block eliminates windows 32x32, undefined and nil errors
  (setf id_list_ori (remove-if #'(lambda (x)
   (or
    (string= (nth 2 x) 'undefined)
    (and
     (equal (nth 7 (cadr x)) 32)
     (equal (nth 9 (cadr x)) 32))
    (equal x nil)))
       id_list_ori))

  (setf id_list_max (sort (compress (sort (copy-list id_list_ori)
  #'(lambda (term1 term2)
      (string<
       (nth 2  term1)
       (nth 2 term2)))))
  #'(lambda (term1 term2)
      (< (nth 0  term1) (nth 0 term2)))))

  (setf maxname (make-pathname :name "max.txt"))
  (with-open-file (str maxname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (format str "Ids = ~A~%~%" id_list_max))

  (setf id_list_best (find-if
      #'(lambda (x)
  (and
   (> (cadr x) 5)
   (not (string= (nth 2 (nth 2 x)) 'undefined))))
      id_list_max))

  (setf bestname (make-pathname :name "best.txt"))
```

```
  (with-open-file (str bestname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (format str "Ids = ~A~%~%" id_list_best))


  (setf idsname (make-pathname :name "ids.txt"))
  (with-open-file (str idsname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (format str "Ids = ~A~%" id_list_ori))

  (setf best_result (find_best id_list_ori))

  (setf best_result (substitute '(1.0) nil
(list sol3 sol4 sol5 sol6 sol7 sol8)))

  (setf pos_best (position (eval
    (cons 'min (mapcar #'car best_result)))
   best_result :key #'car :test #'equal))
  (setf best_result
(cons (+ pos_best 3) (nth pos_best best_result)))

  (with-open-file (str resultname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (format str "~%ALL = ~A~%"
  (list (cons 1 sol1)
(cons 2 sol2)
(cons 3 sol3)
(cons 4 sol4)
(cons 5 sol5)
(cons 6 sol6)
(cons 7 sol7)
(cons 8 sol8))))

  (setf resname (make-pathname :name "res.txt"))
  (with-open-file (str resname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (format str "Best Temp_Id = ~A~%Best_Win__Id = ~A~%"
  best_result id_list_best))
  (format t "Best Temp_Id = ~A~%Best_Win_Id = ~A~%" best_result id_list_best)
  'DONE
)
```

```
;;-------------------------------------------------
;; Main function
;;
;;-------------------------------------------------

(defun recon_all ()
  (setf result_list '())
  (setf resultname (make-pathname :name "recon.txt"))
  (with-open-file (str resultname :direction :output :if-exists :append
       :if-does-not-exist :create))
  (initialize "weights.lisp" "idl3all.lisp" "win_idl3.lisp")
  (scan)
  (if (car best_result)
      (setf result_list (cons best_result result_list)))

;; To be used with multiple NN System
;;  (initialize "weightsb.lisp" "idl3b.lisp" "win_idl3.lisp")
;;  (scan)
;;  (if (car best_result)
;;      (setf result_list (cons best_result result_list)))
;;  (initialize "weightsf.lisp" "idl3f.lisp" "win_idl3.lisp")
;;  (scan)
;;  (if (car best_result)
;;      (setf result_list (cons best_result result_list)))
;;  (initialize "weightsk.lisp" "idl3k.lisp" "win_idl3.lisp")
;;  (scan)
;;  (if (car best_result)
;;      (setf result_list (cons best_result result_list)))

  (if (car result_list)
      (setf best_id (nth (position (eval
    (cons 'min (mapcar #'cadr result_list)))
   result_list :key #'cadr :test #'equal)
 result_list)))
  (with-open-file (str resultname :direction :output :if-exists :append
       :if-does-not-exist :create)
  (if (and best_id (< (cadr best_id) solthres))
      (format str "Best Id = ~A~%"
      best_id)
    (format str "NO RESULTS")))
  (format t "Best Id = ~A~%" best_id)
  'DONE
```

)

APPENDIX F. C CODE FOR EDGE DETECTION

```
/*********************************************************************/
/*   FILENAME: edge_types.h
/*    CHANGES: Jader Gomes da Silva Filho
/*   Last Update: 12/20/96
/*DESCRIPTION: Type definitions for edge detection
/*********************************************************************/



struct point_type
{
  double x,y; /* x,y coordinates of the pixel endpoints */
};
typedef struct point_type POINT;



typedef struct line_type
{
  char name[3];  /* for troubleshooting */

  POINT p1, p2;  /* the 2 endpoints for the line */

  /* Least Squares Fit momments: */

  double  nn; /* changed */

  double m00;    /* Number of pixels should be long*/

  double m10;    /* Sum x */

  double m01;    /* Sum y */

  double m11;    /* Sum x*y */

  double m20;    /* Sum x*x */

  double m02;    /* Sum y*y */

  double phi;    /* Calculated normal orientation of IMG_LINE */

  double dmajor; /* Length of major axis of equivallent ellipse */
```

```
  double dminor; /* Length of minor axis of equivallent ellipse */

  double rho;    /* Ratio dminor/dmajor */

  /* Pattern Matching Information: ------------------------------- */
  double    angle_to_image_center;

  int *matchlist;/* Bogus pointers for IMG_LINE *create_line(EDGE *r,...) */
                 /* in 'edgesupport.c'. */
  int *pm;       /* These pointers are required for matching and are
    declared as (MATCHTYPE *)s in 'match_types.h' */

  struct line_type *next; /* ptr to the next IMG_LINE in the image */

} IMG_LINE;


typedef struct edge_region_type
{
  long   first_pixel;
  long   last_pixel ;
  double avg_phi;
  double sum_phi;
  double  nn;/* changed */
  double  m00 ;
  double  m10 ;
  double  m01 ;
  double  m11 ;
  double  m20 ;
  double  m02;
  char *region_number;
  struct edge_region_type *next ;
} REGION;


 typedef struct pixel_info
 {
  double phi;
  REGION *r;
  int significant;
  char *region_number;
```

```
double mag;/* changed */
} PIXEL ;
```

```
/*************************************************************************/
/*   FILENAME: edgesupport.h
/*     AUTHOR: Khaled Morsy  (some parts from original version by Peterson)
/*    CHANGES: Jader Gomes da Silva Filho
/*       DATE: 11 October 1996
/*   Last Uppdate: 12/20/96
/*
/*   DESCRIPTION: Collection of edge finding functions.
/*
/*   void fatal(char message)
/*   int gradient_angles_close(double r,double s)
/*   int close_to_negative_pi(double phi)
/*   int horizontal(EDGE *r)
/*   EDGE *create_edge(long z,double phi)
/*   void add_pixel_to_edge (long z, double phi, EDGE *r)
/*   EDGE *combine_edges(EDGE *r1, EDGE *r2)
/*   IMG_LINE *create_line (EDGE *r, double M20, double M11, double M02,
/*                      double Dmajor, double Dminor, double Rho)
/*   void line_test (EDGE *r)
/*   void check_active_edges ()
/*   void rgbalong_to_bwlong (long rgbalong, long *bwlong)
/*   void pixel_membership (long z, double phi)
/*   void set_pixel_white (long *rgbalong)
/*   void set_pixel_black (long *rgbalong)
/*   void write_all_lines (long x, long y)
/*   IMG_LINE *fastlines (NPSIMAGE *img)
/*************************************************************************/

/* Gradient Magnitude Threshold - for fastlines(img)                    */
#define THRESHOLD            100.0

/* ANGLE LIMIT SUCH THAT A LINE IS VERICAL IF ITS ORIENTATION EITHER BETWEEN
   -1.8 DEG TO +1.8 DEG OR ( 178.2 DEG TO 181.8 DEG) -- by khaled 5-8-95 */
/*#define VERT_LIMIT           .0314    /* ABOUT PI/100 ~~ 1.8 DEG */
  #define VERT_LIMIT           .0174    /* ABOUT PI/18 ~~ 10 DEG */
/* Gradient Angular Orientation */


 #define MAX_DELTA_PHI   (PI*22/180) /* maximum difference (in radians)*/

/* Constants for function:  void line_test (EDGE *r) */
```

```
#define MIN_PIXELS_PER_LINE  30   /* was 10 minimum pixels allowed for a IMG_LI
#define MIN_DMAJOR        15.0  /*was 10.0 * minimum major axis length allowed
#define MAX_RHO             1.0   /* maximum ratio (Rho=Dminor/Dmajor) */
#define PI                  3.14159265

/* --- Global variables ----------------------------------------------*/
 FILE *reg_file , *image_out ;
 long   Xdim;                       /* width of input image (nr pixels) */
 long   Ydim;                       /* width of input image (nr pixels */
 int    Linecount = 0;              /* counter for number of IMG_LINEs made  */
 int    gray ;                      /* added by khaled 1-8-95 */
 int    reg_num = 0;
 int    reg_count = 0;                        /* added by khaled -- to be dele
 char *image_data[646][486];
 int count = 0;
/* Pointers to: first region , last region and IMG_LINE list*/

REGION *first_region = NULL;
REGION *last_region= NULL  ;

IMG_LINE   *Line_list_head = NULL;
int neighbor_included;
double min;
FILE *reg_file;

/*********************************************************************
   FUNCTION : normalize()
   PURPOSE  : return the normalized value of orientation
*********************************************************************/

double normalize(o)
double o;
{

 int d = 0;
  if(o>=-PI && o <PI) return(o);

  d=(o+PI)/(2*PI)  ;
  if (d>=0 && o > 0.0)
    d=d+1 ;
  o = o -( 2 *PI *(d-1)) ;
```

```
  return (o);
}

/*********************************************************************
   FUNCTION : convert()
   PURPOSE  : return the charater conversion of integer
*********************************************************************/
char *convert(n)
int n;
{
 char *Y[10] = {"0","1", "2", "3" ,  "4" ,  "5" ,  "6",  "7",  "8",  "9"};
  return Y[n];
}

/*********************************************************************/
void update(current,x,y)
PIXEL current[];
long x,y;
{
 REGION *reg;
 double o;

 o=current[x].phi;
 reg=current[x].r;
 reg->last_pixel =  (Xdim * y + x);
 reg->sum_phi+=o;
 ++reg->nn;                              /*added*/
 reg->m00+=current[x].mag;               /*changed*/
 reg->m10+=current[x].mag*x;             /*changed*/
 reg->m01+=current[x].mag*y;             /*changed*/
 reg->m11+=current[x].mag*x*y;           /*changed*/
 reg->m20+=current[x].mag*x*x;           /*changed*/
 reg->m02+=current[x].mag*y*y;           /*changed*/
 reg->avg_phi = reg->sum_phi / reg->nn;/*changed*/

 image_data[x][y] = current[x].region_number ;
/* fprintf(reg_file,"\n%d %d %d %d",x,y,current[x].region_number, reg ); */
 /* printf("\nfirst %d , last %d, avg phi %lf",reg->first_pixel, reg->last_pixel
          ,reg->avg_phi);*/

}
/*********************************************************************/
```

```
void compare1(current,x,y)
PIXEL current[];
long x,y;
{
  double t;
  REGION *current_region;
  t= fabs(normalize(current[x].phi-current[x-1].phi));

  if ((t<= MAX_DELTA_PHI )&&(current[x-1].significant == 1))
   {


     min = t;
     current[x].r = current[x-1].r;
  current[x].region_number = current[x-1].region_number ;
     neighbor_included = 1;

   }

}

/*****************************************************************/
void compare2(current,prev,x1,x2,y)
PIXEL current[],prev[];
long x1,x2,y;
{
  double t;
  REGION *current_region;
  t=fabs(normalize(current[x1].phi - prev[x2].phi));


 if ((t<= MAX_DELTA_PHI )&&(prev[x2].significant == 1))
   {
     if (t<min)
       {
         min = t;
         current[x1].r = prev[x2].r;
current[x1].region_number = prev[x2].region_number ;
 neighbor_included = 1;
       }
   }
```

```
}

/*********************************************************************/


REGION *create_region(x,y,o,mag)
long x,y;
double o,mag;/*added mag to the function call*/
{
 REGION *reg;

 if((reg=(REGION *) malloc(sizeof(REGION)))==NULL)
   printf("Error creating  Region");
 ++reg_count;
 reg->first_pixel = (Xdim * y + x); if( reg->first_pixel < 0) printf("ERROR");
 reg->last_pixel =  (Xdim * y + x);
 reg->avg_phi = o;
 reg->sum_phi = o;
 reg->nn = 1.0;       /*added*/
 reg->m00 = mag;      /*changed*/
 reg->m10 = mag*x;    /*changed*/
 reg->m01 = mag*y;    /*changed*/
 reg->m11 = mag*x*y; /*changed*/
 reg->m20 = mag*x*x; /*changed*/
 reg->m02 = mag*y*y; /*changed*/
 reg->region_number = convert(reg_num);
 reg->next = NULL;

 if(first_region == NULL)
    first_region = reg;
 if(last_region != NULL)
    last_region->next=reg;
 last_region=reg;

 return(reg);
}


/*********************************************************************/
void pixel_membership(current,prev,x,y,o)
PIXEL current[],prev[];
```

```
long x,y;double o;

{
  REGION *reg;
 /* if(y > 484)
    {printf("invalid y ");
  exit(-1);
 }
  */
  if (x > 1)
  {
     compare1(current,x,y);
     if (y > 1)
     {
        compare2(current,prev,x,x-1,y);
     }
  }
  if (y > 1)
  {
     compare2(current,prev,x,x,y);
      compare2(current,prev,x,x+1,y);
  }

        if (neighbor_included == 0)
        {
  /*added mag to the function call*/
  reg =  create_region(x,y,o,current[x].mag);

  count++;
  current[x].region_number = convert(reg_num);
  image_data[x][y] = current[x].region_number ;
  reg_num++ ;
  if (reg_num > 9)
            reg_num=0;
  current[x].r = reg;

}

  if (neighbor_included == 1)
    {
      update(current,x,y);
    }
```

```
  /*printf("%d y = %d",  y);*/

  if(x >  365 && x< 399)

    fprintf(reg_file,"\n%d %d %s",x,y,current[x].region_number);
  /*fprintf(reg_file,"\n%d %d %s",x,y,current[x].r);*/


}

/*-----------------------------------------------------------------*/
/* void fatal (char message)
/*
/*   Prints error message and exits out of the program.
/*-----------------------------------------------------------------*/
fatal(message)
     char *message;
{
  fprintf(stderr, "Fatal ERROR: ");
  perror(message);
  exit(-1);        /* exit by failure */
}

/*-----------------------------------------------------------------*/
/* int close_to_negative_pi (double phi)
/*
/*   Returns 1 if orientation phi is within MAX_DELTA_PHI to -PI.
/*   Returns 0 otherwise.
/*-----------------------------------------------------------------*/
int close_to_negative_pi(phi)
     double phi;
{
  return(PI + phi < MAX_DELTA_PHI);
}


/*-----------------------------------------------------------------*/
/* int horizontal (EDGE *e)
/*
/*   Returns 1 if orientation of EDGE e (r->avg_phi) is within
/*    MAX_DELTA_PHI/4 to PI/2 or -PI/2 (the normal orientations for
/*    a horizontal line).
```

```
/*   Returns 0 otherwise.
/*-------------------------------------------------------------------*/
int horizontal(e)
     REGION *e;
{
  double maxphi = MAX_DELTA_PHI / 4.0;

  return((fabs(e->avg_phi) > (0.5*PI)-maxphi) &&
 (fabs(e->avg_phi) < (0.5*PI)+maxphi));
}


/*-------------------------------------------------------------------*/
/* IMG_LINE *create_line (EDGE *r, double M20, double M11, double M02,
/*                     double Dmajor, double Dminor, double Rho)
/*
/*   Returns pointer to newly instantiated IMG_LINE with variables set
/*   according to moments described by EDGE r, secondary moments
/*   M20, M11, and M02, axis lengths Dmajor, Dminor, and ratio of
/*   axis lengths Rho.
/*-------------------------------------------------------------------*/
IMG_LINE *create_line(r,M20,M11,M02,Dmajor,Dminor,Rho)
     REGION *r;
     double M20,M11,M02,Dmajor,Dminor,Rho;
{
  IMG_LINE   *l;
  long x1, y1, x2, y2;
  double Phi,  delta1,  delta2;
  int negative_phi;
  /* r->first_pixel mapped onto the IMG_LINE will be endpoint p1
     r->last_pixel mapped onto the IMG_LINE will be endpoint p2 */

x1 = r->first_pixel%(Xdim);
         y1 = r->first_pixel/(Xdim);
         x2 = r->last_pixel%(Xdim);
         y2 = r->last_pixel/(Xdim);

  /* Calculate the normal orientation of the IMG_LINE by atan2() function. */
         Phi    = atan2(-2*M11,M02-M20)/2.0;

  /* Delta1 and delta2 are the offsets used to calculate the endpoints
     for the IMG_LINE segment based upon values x1,y1 and x2,y2. */
```

```
        delta1 = (r->m10/r->m00 - (double)x1)*fcos(Phi) +
         (r->m01/r->m00 - (double)y1)*fsin(Phi);
        delta2 = (r->m10/r->m00 - (double)x2)*fcos(Phi) +
         (r->m01/r->m00 - (double)y2)*fsin(Phi);

 /* Phi = atan2(-2*M11,M02-M20)/2.0) always returns positive result to Phi.
    Therefore, negative_phi = (r->avg_phi < 0.0) is necessary. */
negative_phi = (r->avg_phi < 0.0);

 /* Allocate memory for IMG_LINE l. */
 if((l = (IMG_LINE *)malloc(sizeof(IMG_LINE))) == NULL) {
   fatal("create_line: malloc\n");
 }

 /* Calculate x,y coordinates for endpoints p1 and p2. */
 l->p1.x = (double)x1 + delta1*fcos(Phi);
 l->p1.y = (double)y1 + delta1*fsin(Phi);
 l->p2.x = (double)x2 + delta2*fcos(Phi);
 l->p2.y = (double)y2 + delta2*fsin(Phi);

 /* Copy least squares fit moments. */
 l->m00 = r->m00;
 l->m10 = r->m10;
 l->m01 = r->m01;
 l->m11 = r->m11;
 l->m20 = r->m20;
 l->m02 = r->m02;
 /* Phi is positive, but -pi < r->avg_phi < pi. */
 if(negative_phi) l->phi = -Phi;
 else l->phi = Phi;

 /* Update rest of IMG_LINE values. */
 l->next   = NULL;

 l->dmajor = Dmajor;
 l->dminor = Dminor;
 l->rho    = Rho;

 l->angle_to_image_center = 0.0; /* default values for LINE matching */
 l->matchlist  = NULL;
 l->pm         = NULL;
```

```
  ++Linecount;      /* Increment global variable, Linecount. */
  strcpy(l->name,"   ");
  sprintf(l->name, "%d", Linecount);

  return(l);        /* Return IMG_LINE l. */
}



/*-------------------------------------------------------------*/
/* void line_test (EDGE *r)
/*
/*   Determines if EDGE r meets three requirements to be a IMG_LINE:
/*     (1) The number of pixels in EDGE r (r->m00) be greater than
/*         MIN_PIXELS_PER_LINE.
/*     (2) The ratio (Rho) of the length of major and minor axes of the
/*         EDGE be less than MAX_RHO.
/*     (3) The length of the major axis (Dmajor) be greater than
/*         MIN_DMAJOR, the minimum IMG_LINE length allowed.
/*   If all three conditions are met, a new IMG_LINE type is created and
/*   appended to the Line_list in order of significance (in this case
/*   Dmajor).
/*-------------------------------------------------------------*/
line_test(r)
     REGION *r;
{
  IMG_LINE   *l, *insert_pt = Line_list_head;
  double M20,M11,M02,Ma,Mb,Mmajor,Mminor,Dmajor,Dminor,Rho;


/*************************************************************************/
  /* First test -- A IMG_LINE must have a required minimun number of pixels. */
  if(r->m00 > MIN_PIXELS_PER_LINE)
/*************************************************************************/


    {

      /* Calculate secondary moments by least squares fit. */
      M20 = r->m20 - ((r->m10*r->m10)/r->m00);
      M11 = r->m11 - ((r->m10*r->m01)/r->m00);
      M02 = r->m02 - ((r->m01*r->m01)/r->m00);
```

```
      /* Calculate major and minor axis lengths, Dmajor and Dminor. */
      Ma = (M20+M02)/2.0;
      Mb = sqrt( ((M02-M20)*(M02-M20)/4.0) + (M11*M11) );
      Mmajor = Ma - Mb;
      Mminor = Ma + Mb;
      Dmajor = 4.0*sqrt(Mminor/r->m00);
      Dminor = 4.0*sqrt(Mmajor/r->m00);


      /* Calculate ratio Rho. */
      Rho = Dminor/Dmajor;

      /* Second & Third tests -- Ratio Rho must represent a line, not a blob.
                              -- IMG_LINE must be at least a certain length.
      if((Rho < MAX_RHO) && (Dmajor > MIN_DMAJOR))
{
  /* The EDGE passed the three requirments to be a line. */
  l = create_line(r,M20,M11,M02,Dmajor,Dminor,Rho);


  /* Add new IMG_LINE to IMG_LINE list in order by IMG_LINE length, dmajor. */
  if(Line_list_head == NULL)
  {
     Line_list_head = l;
  }
  else if(l->dmajor > Line_list_head->dmajor)
  {
     l->next = Line_list_head;
     Line_list_head = l;


  }
  else
  {
    while((insert_pt->next != NULL) &&
  (l->dmajor < insert_pt->next->dmajor))
    {
   insert_pt = insert_pt->next;
    }
       l->next = insert_pt->next;
       insert_pt->next = l;
```

```
  }

} /* end if second and third tests */
    }     /* end if first test */
}


/*-------------------------------------------------------------*/


/*-------------------------------------------------------------*/
/* void rgbalong_to_bwlong (long rgbalong, long *bwlong)
/*
/*   Converts color to black/white for rgba formatted pixels.
/*   The weights assigned for each color are television standards.
/*   This function courtesy of M. Zyda.
/*-------------------------------------------------------------*/

rgbalong_to_bwlong(rgbalong,bwlong)

  long rgbalong;  /* input color rgbalong */
     long *bwlong;   /* output b/w rgbalong  */

{
  unsigned char red, green, blue, alpha;
  unsigned bw;

  /* Use bit masks to get RGB and alpha values from input rgbalong. */
  red   =  rgbalong & 0x000000ff;
  green = (rgbalong & 0x0000ff00) >> 8;
  blue  = (rgbalong & 0x00ff0000) >> 16;
  alpha = (rgbalong & 0xff000000) >> 24;
/*  if ( i== 4000  || i== 5000) printf("\n%d  :alpha = %d" , i, alpha);*/
  /* Calculate the black&white intesity using NTSC standard.
     intensity = 0.299(red) + 0.587(green) + 0.114(blue)        */
  bw = (0.299*red) + (0.587*green) + (0.114*blue);

  /* Save the black&white intensity in bwlong. */
  *bwlong = (alpha<<24)|(bw<<16)|(bw<<8)|bw;
  gray=bw ;

}
```

```
/*-------------------------------------------------------------------*/
/* void set_pixel_white (long *rgbalong)
/*
/*   Sets the specified long integer pointed to by rgbalong to be
/*   white by setting all RGB bits to ff (255 dec -> max intensity).
/*-------------------------------------------------------------------*/
set_pixel_white(rgbalong)
     long *rgbalong;
{
  *rgbalong = 0xffffffff;
}


/*-------------------------------------------------------------------*/
/* void set_pixel_black (long *rgbalong)
/*
/*   Sets the specified long integer pointed to by rgbalong to be
/*   black by setting all RGB bits to 00 (0 dec -> min intensity).
/*-------------------------------------------------------------------*/
set_pixel_black(rgbalong)
     long *rgbalong;
{
  *rgbalong = 0xff000000;
}


/*-------------------------------------------------------------------*/
/*  void write_all_lines (long x, long y)
/*
/*    Write to output file "name.text".
/*-------------------------------------------------------------------*/
write_all_lines(datafile,x,y)
     char *datafile; /*added*/
     long x,y;       /* the x,y dimensions of the image */
{
  IMG_LINE *l = Line_list_head;
  FILE *lines_file;

  lines_file = fopen("lines.text","w");
  fprintf(lines_file,"--- DATA FOR EXTRACTED LINE SEGMENTS ---\n");
```

```
  fprintf(lines_file,"Image size: nr pixels x axis = %d, nr pixels y axis =  %d
  fprintf(lines_file,"Extracted line segments listed in order by length.\n\n");

  /*added for printing number of lines and regions*/
  fprintf(lines_file,"\nNumber lines found in %s = %d", datafile,Linecount);
  fprintf(lines_file,"\nNumber regions found in %s = %d\n\n", datafile,reg_coun

  while(l!=NULL)
    {

      /*changed to show m00 as double*/
      fprintf(lines_file,"%s> length = %.4f, thinness = %.4f, orientation = %.4
      l->name, l->dmajor, l->rho, l->phi, l->m00);

      fprintf(lines_file,"endpoints: (%.2f %.2f)  (%.2f %.2f)\n\n",
      l->p1.x,l->p1.y,l->p2.x,l->p2.y);
      l = l->next;
    }
  fclose(lines_file);

  printf("  lines found in image written to: 'lines.text'\n");
}


/*********************************************************************/
/*  int vertical_edge(EDGE)
/*   return 1 if orientation of an edge is
     (BETWEEN -VERT_LIMIT & +VERT_LIMIT) OR (> PI - VERT_LIMIT )
  OR ( < - (PI + VERT_LIMIT))                                    */

/*********************************************************************/
int vertical_edge(r)
  REGION *r;
{
   return ((fabs(r->avg_phi) < VERT_LIMIT) ||
        (fabs(r->avg_phi) > PI - VERT_LIMIT)) ;

}
```

```
/*****************************************************************/
/*   FILENAME: findedge.c
/*     AUTHOR: Khaled Morsy  & L. Remias
/*    CHANGES: Jader Gomes da Silva Filho
/*       DATE: 18 January 1996
/*   Last Update: 12/20/96
/*DESCRIPTION: An image gradient program incorporating edge-finding.
/*        Displays edge-gradient image and associated lines.
/*
/*             This application is designed for use on a Silicon-
/*             Graphics Iris (r) workstation utilizing a .sgi
/*             or similar rgb formatted image.
/*             RGB values are of type LONG in the form AABBGGRR where:
/*               AA - alpha value, 0-255
/*               BB - blue component, 0-255
/*               GG - green component, 0-255
/*               RR - red component, 0-255
/*
/*             SiliconGraphics graphics library functions used within
/*             the display_bw_and_gradient_images() routine:
/*               qdevice(), winset(), c3f(), move2(), draw2(),
/*               swapbuffers(), reshapeviewport(), winclose(),
/*               and lrectwrite().
/*
/*             NPSIMAGE function routines borrowed courtesy of M.Zyda:
/*               read_sgi_rgbimage(), get_empty_rgba_npsimage(),
/*               get_empty_rgb_npsimage(), and rgbalong_to_bwlong().
/*
/*        Least Squares Fit method for line-finding from
/*        "Sonar Data Interpretaion for Autonomous Mobile Robots"
/*        by Y.Kanayama, T.Noguchi, & B.Hartman, 1990.
/*****************************************************************/
#include <gl.h>              /* SiliconGraphics (r) graphic library   */
#include <gl/image.h>        /* SGI image structure library           */
#include <device.h>          /* Machine-dependent device library      */
     /* for keys and mouse-buttons           */
#include <stdio.h>           /* C standard i/o library                */
#include <math.h>            /* C math library for atan2()            */

#include "image_types.h"     /* Type definitions for NPSIMAGE, etc.   */
#include "edge_types.h"      /* Type definitions for EDGE, LINE, etc  */
#include "npsimagesupport.h" /* Some NPSIMAGE functions               */
```

```
/* #include "edgesupport.h"
    /* EDGE and IMG_LINE building functions  */
#include "esworkalso1.h"
/*#include "es96.h"*/
#include "displaysupport.h"  /* Graphics display functions            */

int neighbor_included;

main(argc, argv)
     int argc;
     char *argv[];
{
  NPSIMAGE *img1, /* input file_name.rgb color image */
           *img2, /* black&white image */
           *img3; /* gradient image */

  /* pointers to RGBA longs ("bitsptr"s) of respective NPSIMAGEs */
  long     *ptr1, *ptr2, *ptr3;
  double    dx, dy, Th = THRESHOLD*THRESHOLD;
  int z = 0;  /*counter for pixels in gradient image */
  REGION *reg;

  /*****added 950830 km lr*****************************************/
PIXEL current[646],prev[646];
double orientation;
int UL,U,UR,L,R,DL,D,DR,c;
int i,s,t,p;
long     x, y;
long  ptr4[313956];  /*646 x 486 */
reg_file = fopen("regions.txt","w");
image_out = fopen("image_out.dat", "w");

/*********************************************/

  if(argc != 2) fatal("usage: findedge filename\n");

  /* Read in input rgb image */
  img1 = read_sgi_rgbimage(argv[1]);
  if(img1 ==(NPSIMAGE *) NULL)
    {
      fatal("File %s is a NULL image.\n",img1->name);
    }
```

```
 Xdim = img1->xsize;                /* else set global Xdim and ptr1 */
 Ydim = img1->ysize ;

 ptr1 = img1->imgdata.bitsptr;
/* printf("findedge:> %s xsize= %d ysize= %d pixels= %d\n",
  img1->name,img1->xsize,img1->ysize, (img1->xsize*img1->ysize));*/


  /* Declare new NPSIMAGEs */
 if((img1->type == RGBAWITHALPHA) || (img1->type == RGBA))
   {
     img2 = get_empty_rgba_npsimage(Xdim,img1->ysize,img1->name);
     img3 = get_empty_rgba_npsimage((Xdim-2),img1->ysize-2,"findedge");
   }
 else
   {
     fatal("Unknown or c-mapped image type: %d.\n",img1->type);
   }
 ptr2 = img2->imgdata.bitsptr;
 ptr3 = img3->imgdata.bitsptr;

 /* The scan of an RGB image is from the bottom row -> up,
    traversing the rows left to right. */

 /* In order for the Sobel operator to be calculated for a specific pixel,
    all eight surronding pixels must have an absolute (black & white)
    light intensity calculated.  Function rgbalong_to_bwlong() performs
    this task. */

 /* Due to the nature of the Sobel operator, the pixels in the top and
    bottom rows as well as pixels in the leftmost and rightmost columns
    of the input image will not be calculated.  In order to start cal-
    culating the Sobel operators, the first 2 rows of the input image
    must be converted to black & white light intensity values. */

 /* Calculate bw values for the entire input image. */
 for(i=0; i< ((Xdim * Ydim)-1);++i)
   {
     rgbalong_to_bwlong( ptr1[i],&ptr2[i]);
  ptr4[i] = gray;
}
```

```
/*****************************************************************/
/*  modified by Khaled & Remias ***********************************/

 for (y=0; y < Ydim; ++y)
 {
   for (x=0; x < Xdim; ++x)
   {
    i = (y * Xdim) + x;
       neighbor_included = 0;
       min = PI;

 UL=i+(Xdim-1); U = UL+1 ; UR = U+1;
       L= i-1 ; R = i+1;
       DL=i-(Xdim-1); D=DL+1; DR = D+1;

if(x == (Xdim-1))
   {
 for (c=0; c<(Xdim-1); c++)
    {
            prev[c]=current[c];
            current[c].phi=0.0;
            current[c].r = NULL;
            current[c].significant = 0;
       }
   }

if((y == 0) || (y == (Ydim-1)) || (x == 0) || (x == (Xdim-1)))

   {
   set_pixel_black(&ptr3[i]);
   }

else
{
  /* Calculate dx,dy via Sobel operator for pixel i. */
  dx = - (ptr4[i+(Xdim-1)]) + (ptr4[i+(Xdim+1)])
        -  2 *( ptr4[i-1])+ 2 *(ptr4[i+1])
 - (ptr4[i-Xdim-1]) + (ptr4[i-Xdim+1]);
          dy =  (ptr4[i+Xdim-1]) + 2*(ptr4[i+Xdim]) +(ptr4[i+Xdim+1])
  -(ptr4[i-Xdim-1]) - 2*(ptr4[i-Xdim]) - (ptr4[i-Xdim+1]);
```

```
  if( ((dx*dx)+(dy*dy)) > Th)
  {
    orientation = atan2(dy,dx);
    current[x].phi = orientation;
 current[x].mag = sqrt((dx*dx)+(dy*dy)); /* changed */
/*printf("I am here %f \n",current[x].mag);*/
            current[x].significant = 1;
            set_pixel_black(&ptr3[z]);

       pixel_membership(current,prev,x,y,orientation);

 /* printf("\n %d %d %s",x, y,  image_data[x][y]);*/
         }

        else
           {  image_data[x][y] = ".";
              set_pixel_white(&ptr3[z]);
           }
        ++z;
}
    } /* endfor x */
  } /* end for y */

  fclose(reg_file);

    reg = first_region;

    while (reg != NULL)
    {
 /* printf("\nreg%u m00=%f", reg,reg->m00);*/
 /* printf("\nm00=%f m10=%f m01=%f m11=%f m20=%f m02=%f",reg->m00,
 reg->m10,reg->m01,reg->m11,reg->m20,reg->m02);*/
 line_test(reg);
 reg = reg->next;
    }

  /* Write the lines list to file "lines.text". */
  write_all_lines(argv[1],img3->xsize,img3->ysize);
  printf("\nNumber lines found in %s = %d", argv[1],Linecount);
  printf("\nNumber regions found in %s = %d\n", argv[1],reg_count);
  /*************************************************************/
  /*************** VISUAL DISPLAY OF REGIONS .. BY KHALED MORSY   */
```

```
  /*****************************************************************/
  s=1;
  t=91;
 for(count=1;count<8;++count)
   {
       for(y=Ydim-2; y >1 ; --y)
       {
          for(x=s; x < t ; ++x)
          {
            p = (x/90) + ((y/63) * 9)+1 ;
            /*** we can put any number (t) in place of 3 to print col # t */
            if((p%9)==5&& image_data[x][y] != NULL)
               { if(x==s)
           fprintf(image_out ,  "\n");

fprintf(image_out, "%s",image_data[x][y]);
       }

          }
        }
        s=t; t+=90;  fprintf(image_out, "\n \n");
   }
  /*****************************************************************/


  /* Display the black&white and gradient images on the screen. */
  display_bw_and_gradient_images(img2,img3,Line_list_head);
  display_line_image(img2,Line_list_head);
  printf("\nfindedge %s...done.\n",argv[1]);
}
```

APPENDIX G. LISP CODE FOR GENERATING A 3D MODEL

```
; ************************************************************************
; File: euler-angle-rigid-body.lisp    Franz Common LISP
;
; File that defines the class "rigid-body" and its methods.
;
; by Prof. Robert McGhee
; Modified by Jader Filho -- 1997
; ************************************************************************

(load "robot-kinematics")

(defclass rigid-body
    ()
  ((posture                ;The vector (xe ye ze phi theta psi).
    :initform '(0 0 0 0 0 0)
    :initarg :posture
    :accessor posture)
   (posture-rate ;The vector (xe-dot ye-dot ze-dot phi-dot theta-dot psi-dot).
    :initarg :posture-rate
    :accessor posture-rate)
   (velocity               ;The six-vector (u v w p q r) in body coordinates.
    :initform '(1 1 1 .1 .1 .1)
    :initarg :velocity
    :accessor velocity)
   (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
    :accessor velocity-growth-rate)
   (forces-and-torques    ;The vector (Fx Fy Fz L M N) in body coordinates.
    :initform (list 0 0 (- *gravity*) 0 0 0)
    :accessor forces-and-torques)
   (moments-of-inertia    ;The vector (Ix Iy Iz) in principal axis coordinates.
    :initform '(1 1 1)
    :initarg :moments-of-inertia
    :accessor moments-of-inertia)
   (mass
    :initform 1
    :initarg :mass
    :accessor mass)
   (node-list    ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
   :initform '((0 0 0 1) (0 10 0 1))
    :initarg :node-list
    :accessor node-list)
    (polygon-list
```

```
     :initform '((0 1))
     :initarg :polygon-list
     :accessor polygon-list)
    (transformed-node-list  ;(x y z 1) in earth coord for each node in node-list.
     :accessor transformed-node-list)
    (H-matrix
     :initform (unit-matrix 4)
     :accessor H-matrix)
    (time-stamp
     :accessor time-stamp)))

(defmethod initialize ((body rigid-body))
  (setf (transformed-node-list body) (node-list body))
  (setf (velocity-growth-rate body) (update-velocity-growth-rate body))
  (setf (posture-rate body) (earth-velocity body))
  (setf (time-stamp body) (get-internal-real-time)))

(defmethod move ((body rigid-body) azimuth elevation roll x y z)
  (setf (posture body) (list x y z roll elevation azimuth))
  (setf (H-matrix body)
    (homogeneous-transform azimuth elevation roll x y z))
  (transform-node-list body))

(defmethod get-delta-t ((body rigid-body)) 0.1)
;  (let* ((new-time (get-internal-real-time))
;         (delta-t (/ (- new-time (time-stamp body)) 1000)))
;   (setf (time-stamp body) new-time)
;   delta-t))                              .

(defmethod update-rigid-body ((body rigid-body))     ;Euler integration.
  (let* ((delta-t (get-delta-t body)))
       (update-posture body delta-t)
       (setf (H-matrix body) (homogeneous-transform (sixth (posture body))
          (fifth (posture body)) (fourth (posture body)) (first (posture body))
          (second (posture body)) (third (posture body))))
       (transform-node-list body)
       (update-velocity body delta-t)
       (update-velocity-growth-rate body)))

(defmethod update-velocity-growth-rate ((body rigid-body))
  (setf (velocity-growth-rate body) ;Assumes principal axis coordinates with
    (multiple-value-bind            ;origin at center of gravity of body.
```

```
      (Fx Fy Fz L M N   u v w p q r   Ix Iy Iz) ;Declares local variables.
      (values-list                              ;Values assigned.
        (append
          (forces-and-torques body) (velocity body) (moments-of-inertia body))
      (list (+ (* v r) (* -1 w q) (/ Fx (mass body))
               (* *gravity* (first (third (H-matrix body)))))
            (+ (* w p) (* -1 u r) (/ Fy (mass body))
               (* *gravity* (second (third (H-matrix body)))))
            (+ (* u q) (* -1 v p) (/ Fz (mass body))
               (* *gravity* (third (third (H-matrix body)))))
            (/ (+ (* (- Iy Iz) q r) L) Ix)
            (/ (+ (* (- Iz Ix) r p) M) Iy)
            (/ (+ (* (- Ix Iy) p q) N) Iz)))))  ;Value returned.

(defmethod update-velocity ((body rigid-body) delta-t) ;Euler integration.
  (setf (velocity body)
    (vector-add (velocity body)
                (scalar-multiply delta-t (velocity-growth-rate body)))))

(defmethod update-posture ((body rigid-body) delta-t) ;Euler integration.
  (setf (posture-rate body) (earth-velocity body))
  (setf (posture body)
    (vector-add (posture body) (scalar-multiply delta-t (posture-rate body)))))

(defmethod transform-node-list ((body rigid-body))
  (setf (transformed-node-list body)
    (mapcar #'(lambda (node-location)
                (post-multiply (H-matrix body) node-location))
            (node-list body))))

(defconstant *gravity* 32.2185)

(defmethod earth-velocity ((body rigid-body))
  (let* ((linear-velocity (firstn 3 (velocity body)))
         (rotational-velocity (cdddr (velocity body)))
         (posture (posture body))
         (R-matrix (rotation-matrix (sixth posture) (fifth posture)
                                    (fourth posture)))
         (linear-earth-velocity (post-multiply R-matrix linear-velocity))
         (T-matrix (body-rate-to-euler-rate-matrix (sixth posture)
                     (fifth posture) (fourth posture)))
         (rotational-earth-velocity (post-multiply T-matrix
```

```
                                                    rotational-velocity)))
    (append linear-earth-velocity rotational-earth-velocity)))
```

```
; ************************************************************************
; File: video-camera.lisp    Franz Common LISP
;
; ** VIDEO-CAMERA CLASS DEFINITION **
; A Video-camera is a camera which uses double buffering in order to
; display a sequence of images without flicker.
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
;
; Requires: camera.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Adapted from Prof. Kwak's Movie Camera flavor.
; Modified by Jader Filho -- 1997
; ************************************************************************

(load "camera")
(load "loadfloor")

(defclass video-camera (camera)
   ((image-window
    :accessor image-window
    :initform (cw:make-bitmap-stream :borders 5
                                     :width 300
                                     :height 400
                                     :title "2nd Floor"
                                     :background-color cw:blue
                                     :foreground-color cw:white
                                     :activate-p nil))))

(defun create-video-camera-1 ()
  (setf *camera-1* (make-instance 'video-camera))
  (queue-mouse *camera-1*))

(defmethod new-picture ((camera video-camera) (body rigid-body) draw-color)
  (erase-image-window camera)
  (take-picture camera body draw-color)
  (expose-image camera))


;;modified
(defmethod draw-line-in-window ((camera video-camera) start end draw-color)
```

```
  (let ((line-color (cond
     ((= 0 draw-color) cw:white)
     ((= 1 draw-color) cw:yellow)
     ((= 2 draw-color) cw:magenta)
     ((= 3 draw-color) cw:green)
     ((= 4 draw-color) cw:red)
     ((= 5 draw-color) cw:cyan)
     ((= 6 draw-color) cw:black)
     ((= 7 draw-color) cw:blue))))
    (cw:draw-line (image-window camera)
  (cw:make-position :x (first start) :y (second start))
  (cw:make-position :x (first end) :y (second end))
  :brush-width 0 :color line-color)))


(defmethod expose-image ((camera video-camera))
  (cw:bitblt (image-window camera) 0 0 (camera-window camera) 0 0))

(defmethod erase-image-window ((camera video-camera))
  (cw:clear (image-window camera)))

(defun test-video-camera (az el roll x y z);Produces top view of default rigid-bod
  (reset-windows)
  (create-video-camera-1)
  (loadfloor)
  (initialize  *floor*)
  (setf (posture *camera-1*) (list az el roll x y z)
(H-matrix *camera-1*) (homogeneous-transform-matrix
       (posture *camera-1*))
(inverse-H-matrix *camera-1*) (inverse-H (H-matrix *camera-1*)))
  (move-camera *camera-1* az el roll x y z)
  (new-picture *camera-1* *floor* *floor-color*))

(defun test0 () (test-video-camera (/ pi 2) 0 0 -150 0 -100))

(defun test1 () (test-video-camera (- (/ pi 2)) 0 0 -150 200 -100))

(defun test2 () (test-video-camera pi 0 0 -150 100 -100))

(defun test3 () (test-video-camera pi 0 0 -150 100 -100))
```

```
; ***************************************************************************
; File: movie-camera.lisp   Franz Common LISP
;
; ** MOVIE-CAMERA CLASS DEFINITION **
; A Movie-camera is a video-camera which is able to record and play back
; images. It has film (defined in film.cl) which stores the images
; and a projector (defined in projector.cl) which allows convenient playback
; of the images via a Graphical User Interface (GUI).
;
; Requires: camera.cl video-camera.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Modifications & enhancements to Mark Kindl's Improved-Movie-Camera flavor.
; Modified by Jader Filho -- 1997
; ***************************************************************************

(load "video-camera")
(load "film")


(defclass movie-camera (video-camera)
  ((film ; class containing stored images (bitmaps)
    :initform (make-instance 'film :frames (make-array frames-max))
    :accessor film)
   (projector ; GUI class for convenient film playback
    :accessor projector
    :initform (make-instance 'projector))))

(load "projector")

(defconstant frames-max 60)
(defconstant repetitions 1)

(defun create-movie-camera-1 ()
  (setf *camera-1* (make-instance 'movie-camera))
  (queue-mouse *camera-1*)
  (initialize-film (film *camera-1*))
  (initialize-projector (projector *camera-1*)))

; Displays current image, records image on film, and advances film frame.
(defmethod expose-image ((movie-camera movie-camera)) ;display, record & adv fr
  (cw:bitblt (image-window movie-camera) 0 0 (camera-window movie-camerá) 0 0 )
```

```
  (cw:bitblt (image-window movie-camera) 0 0
     (aref (frames (film movie-camera)) (current-frame (film movie-camera))) 0 0)
  (advance-new-frame (film movie-camera)))


(setf *floor* (make-instance 'rigid-body))

(setf *floor-color* *white*)

(defun test-movie-camera (az el roll x y z);Produces top view of default rigid-bod
  (reset-windows)
  (create-movie-camera-1)
  (loadfloor)
  (initialize  *floor*)
  (setf (posture *camera-1*) (list az el roll x y z)
(H-matrix *camera-1*) (homogeneous-transform-matrix (posture *camera-1*))
(inverse-H-matrix *camera-1*) (inverse-H (H-matrix *camera-1*)))
  (move-camera *camera-1* az el roll x y z)
  (new-picture *camera-1* *floor* *floor-color*))

(defun test4 () (test-movie-camera pi 0 0 -150 100 -100))
```

```
; ************************************************************************
; File: film.lisp         Franz Common LISP
;
; ** FILM CLASS DEFINITION **
; Film is the part of a Movie-camera which stores the recorded images.
;
; Requires: camera.cl video-camera.cl movie-camera.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Modifications & enhancements to Mark Kindl's Improved-Movie-Camera flavor
; Modified by Jader Filho -- 1997
; ************************************************************************


(defclass film ()
  ((current-frame
    :initform 0
    :accessor current-frame)
   (last-frame
    :initform -1
    :accessor last-frame)
   (max-frames
    :initform frames-max
    :accessor max-frames)
   (frames        ; array of stored images (bitmaps)
    :initarg :frames
    :accessor frames)))


(defmethod initialize-film ((film film))
  (dotimes (count (max-frames film))
     (setf (aref (frames film) count)
        (cw:make-bitmap         :bits-per-pixel 8  ; color images
                                :width 300
                                :height 400))))


(defmethod goto-frame (frame-number (film film))
  (setf (current-frame film) frame-number))

(defmethod advance-frame ((film film))  ;Cycle 0 <= current-frame <= last-fram
  (setf (current-frame film) (mod (1+ (current-frame film))
```

```
    (1+ (last-frame film)))))

(defmethod advance-new-frame ((film film))
  ; case where new exposures made over previously stored frames
;  (setf (current-frame film) (mod (1+ (current-frame film))
;     (max-frames film)))  ;Cycle 0 <= current-frame <= max-frames
  ;last-frame <= max-frames -1
  (if (and (< (last-frame film) (1- (max-frames film)))
   (= 1 (- (current-frame film) (last-frame film))))
    (setf (last-frame film) (1+ (last-frame film))))
  (setf (current-frame film) (mod (1+ (current-frame film))
     (max-frames film))))  ;Cycle 0 <= current-frame <= max-frames

(defmethod go-back-frame ((film film)) ;Cycle last-frame >= current-frame >= 0
  (setf (current-frame film) (mod (1- (current-frame film))
     (1+ (last-frame film)))))
```

```
; ***************************************************************************
; File: loadfloor.lisp    Franz Common LISP
;
; File with the data necessary for building a 3D model of a Hallway
;
; by Jader Filho -- 1997
; ***************************************************************************

 (defun loadfloor()
  (setf (node-list *floor*)

'((0.0 0.0 0.0 1)
  (0.0 0.0 0.0 1)
  (0.0 248.5 0.0 1)
  (-164.7 248.5 0.0 1);;door
  (-164.7 256.75 0.0 1)
  (-245.7 256.75 0.0 1)
  (-245.7 248.5 0.0 1)
  (-390.2 248.5 0.0 1)
  (-390.2 256.75 0.0 1)
  (-491.2 256.75 0.0 1)
  (-491.2 248.5 0.0 1)
  (-657.8 248.5 0.0 1)
  (-657.8 845.5 0.0 1)
  (-893.5 845.5 0.0 1)
  (-893.5 771.9 0.0 1)
  (-956.5 771.9 0.0 1)
  (-956.5 818.9 0.0 1)
  (-1096.68 818.9 0.0 1)
  (-1096.68 588.9 0.0 1)
  (-956.5  588.9 0.0 1)
  (-956.5 635.9 0.0 1)
  (-893.5 635.9 0.0 1)
  (-893.5 477.4 0.0 1)
     (-956.5 477.4 0.0 1)
     (-956.5 524.7 0.0 1)
     (-1096.68 524.7 0.0 1)
     (-1096.68 294.7 0.0 1)
     (-956.5 294.7 0.0 1)
     (-956.5 341.7 0.0 1)
     (-893.5 341.7 0.0 1)
     (-893.5 248.5 0.0 1)
```

```
(-1190.5 248.5 0.0 1)
(-1190.5 256.75 0.0 1)
(-1281.5 256.75 0.0 1)
(-1281.5 248.5 0.0 1)
(-1592.9 248.5 0.0 1)
(-1592.9 256.75 0.0 1)
(-1683.9 256.75 0.0 1)
(-1683.9 248.5 0.0 1)
(-2159.3 248.5 0.0 1)
(-2159.3 256.75 0.0 1)
(-2250.3 256.75 0.0 1)
(-2250.3 248.5 0.0 1)
(-2587.3 248.5 0.0 1)
(-2587.3 256.75 0.0 1)
(-2678.3 256.75 0.0 1)
(-2678.3 248.5 0.0 1)
(-3692.0 248.5 0.0 1)
(-3692.0 457.0 0.0 1)
(-3683.75 457.0 0.0 1)
(-3683.75 548.0 0.0 1)
(-3692.0 548.0 0.0 1)
(-3692.0 727.3 0.0 1)
(-3711.6 727.3 0.0 1)
(-3711.6 809.7 0.0 1)
(-3962.2 809.7 0.0 1)
(-3962.2 358.9 0.0 1)
(-3970.45 358.9 0.0 1)
(-3970.45 267.9 0.0 1)
(-3962.2 267.9 0.0 1)
(-3962.2 248.5 0.0 1);;;;
(-5269.7 248.5 0.0 1)
(-5269.7  256.75 0.0 1)
(-5360.7 256.75 0.0 1)
(-5360.7 248.5 0.0 1)
(-5680.7 248.5 0.0 1)
(-5680.7 375.0 0.0 1)
(-5819.2 375.0 0.0 1)
(-5819.2 248.5 0.0 1)
(-6239.2 248.5 0.0 1)
(-6239.2 0.0 0.0 1)
(-5932.7 0.0 0.0 1)
(-5932.7 -8.25 0.0 1)
```

```
(-5841.7 -8.25 0.0 1)
(-5841.7 0.0 0.0 1)
(-5364.2 0.0 0.0 1)
(-5364.2 -8.25 0.0 1)
(-5273.2 -8.25 0.0 1)
(-5273.2 0.0 0.0 1)
(-4796.2 0.0 0.0 1)
(-4796.2 -8.25 0.0 1)
(-4715.7 -8.25 0.0 1)
(-4715.7 0.0 0.0 1)
(-4534.0 0.0 0.0 1)
(-4534.0 -48.0 0.0 1)
(-4447.9 -48.0 0.0 1)
(-4447.9 0.0 0.0 1)
(-4367.9 0.0 0.0 1)
(-4367.9 -8.25 0.0 1)
(-4287.4 -8.25 0.0 1)
(-4287.4 0.0 0.0 1)
(-4228.4 0.0 0.0 1)
(-4228.4 -8.25 0.0 1)
(-4046.4 -8.25 0.0 1)
(-4046.4 0.0 0.0 1)
(-3999.8 0.0 0.0 1)
(-3999.8 -8.25 0.0 1)
(-3919.3 -8.25 0.0 1)
(-3919.3 0.0 0.0 1)
(-3351.3 0.0 0.0 1)
(-3351.3 -8.25 0.0 1)
(-3260.3 -8.25 0.0 1)
(-3260.3 0.0   0.0 1)
(-3018.8 0.0   0.0 1)
(-3018.8 -13.25 0.0 1)
(-3114.0 -13.25 0.0 1)
(-3114.0 -89.25 0.0 1)
(-3131.4 -89.25 0.0 1)
(-3131.4 -609.85 0.0 1)
(-2576.0 -609.85  0.0 1)
(-2576.0 -201.65 0.0 1)
(-2565.0 -201.65 0.0 1)
(-2565.0 -609.85 0.0 1)
(-2014.0 -609.85 0.0 1)
(-2014.0 -89.25 0.0 1)
```

```
(-2032.0 -89.25 0.0 1)
(-2032.0 -13.25 0.0 1)
(-2032.0 0.0   0.0 1)
(-1971.4 0.0 0.0 1)
(-1971.4 -8.25 0.0 1)
(-1880.4 -8.25 0.0 1)
(-1880.4 0.0 0.0 1)
(-1553.2 0.0 0.0 1)
(-1553.2 -8.25 0.0 1)
(-1462.2 -8.25 0.0 1)
(-1462.2 0.0 0.0 1)
(-1311.9 0.0 0.0 1)
(-1311.9 -8.25 0.0 1)
(-1220.9 -8.25 0.0 1)
(-1220.9 0.0 0.0 1)
(-834.4 0.0 0.0 1)
(-834.4 -8.25 0.0 1)
(-743.4 -8.25 0.0 1)
(-743.4 0.0 0.0 1)
(-417.1 0.0 0.0 1)
(-417.1 -8.25 0.0 1)
(-326.1 -8.25 0.0 1)
(-326.1 0.0 0.0 1)
(-265.9 0.0 0.0 1)
(-265.9 -8.25 0.0 1)
(-174.9 -8.25 0.0 1)
(-174.9 0.0 0.0 1)

;;drinking fountain
(-4516.9 -31.0 0.0 1)
(-4516.9 0.0 0.0 1)
(-4485.9 0.0 0.0 1)
(-4485.9 -31.0 0.0 1)

;;island not being use look at the end
(-2123.0 -13.25 0.0 1);;146
(-2508.0 -13.25 0.0 1)
(-2565.0 -13.25 0.0 1)
(-2565.0 -109.25 0.0 1)
(-2576.0 -109.25 0.0 1)
(-2576.0 -89.25 0.0 1)
(-2596.0 -89.25 0.0 1)
```

```
(-2596.0 -13.25 0.0 1)
(-2927.0 -13.25 0.0 1)
(-2927.0 0.0 0.0 1)
(-2123.0 0.0 0.0 1)

;;box1
(-2729.2 -491.85 0.0 1);;157
(-2973.2 -491.85 0.0 1)
(-2973.2 -427.85 0.0 1)
(-2729.2 -427.85 0.0 1)

;;box2
(-2729.2 -276.85 0.0 1);;161
(-2973.2 -276.85 0.0 1)
(-2973.2 -152.65 0.0 1)
(-2729.2 -152.65 0.0 1)

;;ceiling1
(-0.0 0.0 -217.5 1);;165
(-0.0 248.5 -217.5 1);; --2
(-6239.2 248.5 -217.5 1);;--69
(-6239.2 0.0 -217.5 1)
(-507.1 0.0 -217.5 1)
(-507.1 0.0 -257 1)
(-484.6 0.0 -257 1)
(-484.6 0.0 -217.5 1)

;;ceiling2
(-2.5 2.5 -217.5 1);;173
(-2.5 246.0 -217.5 1);;
(-6236.7 246.0 -217.5 1);;
(-6236.7 2.5 -217.5 1)
(-507.1 2.5 -217.5 1)
(-507.1 2.5 -257 1)
(-507.1 0.0 -257 1)
(-484.6 0.0 -257 1)
(-484.6 2.5 -257 1)
(-484.6 2.5 -217.5 1)

;;ceiling3
(-0.0 0.0 -257 1);;183 --173
(-0.0 248.5 -257  1);; --174
```

```
(-6239.2 248.5 -257  1);;--175
(-6239.2 0.0 -257  1);; --176

;;scrub board
(0.0 0.0 -10.0 1)  ;;187
(0.0 43.0 -10.0 1)
(0.0 43.0 0.0 1)
(0.0 0.0 0.0 1)

(0.0 205.5 -10.0 1);;191
(0.0 248.5 -10.0 1)
(-159.7 248.5 -10.0 1)
(-159.7 248.5 0.0 1)
(0.0 248.5 0.0 1)
(0.0 205.5 0.0 1)


(-250.7 248.5 -10.0 1);;197
(-385.2 248.5 -10.0 1)
(-385.2 248.5 0.0 1)
(-250.7 248.5 0.0 1)

(-496.2 248.5 -10.0 1);;201
(-657.8 248.5 -10.0 1)
(-657.8 845.5 -10.0 1)
(-657.8 845.5 0.0 1)
(-657.8 248.5 0.0 1)
(-496.2 248.5 0.0 1)

(-820.3 845.5 -10.0 1);;207
(-893.5 845.5 -10.0 1)
(-893.5 778.9 -10.0 1) ;;+7 elevator
(-893.5 778.9 0.0 1)
(-893.5 845.5 0.0 1)
(-820.5 845.5 0.0 1)

(-893.5 628.9 -10.0 1);;213 -7 elevator
(-893.5 484.4 -10.0 1);;+7 elevator
(-893.5 484.4 0.0 1)
(-893.5 628.9 0.0 1)

(-893.5 334.7 -10.0 1);;217 -7 elevator
```

```
(-893.5 248.5 -10.0 1)
(-1185.5 248.5 -10.0 1)
(-1185.5 248.5 0.0 1)
(-893.5 248.5 0.0 1)
(-893.5 334.7 0.0 1)

(-1286.5 248.5 -10.0 1);;223 +5
(-1587.9 248.5 -10.0 1);;-5
(-1587.9 248.5 0.0 1)
(-1286.5 248.5 0.0 1)

(-1688.9 248.5 -10.0 1);;227
(-2154.3 248.5 -10.0 1)
(-2154.3 248.5 0.0 1)
(-1688.9 248.5 0.0 1)

(-2255.3 248.5 -10.0 1);;231
(-2582.3 248.5 -10.0 1)
(-2582.3 248.5 0.0 1)
(-2255.3 248.5 0.0 1)

(-2683.3 248.5 -10.0 1);;235
(-3276.8 248.5 -10.0 1)
(-3276.8 248.5 0.0 1)
(-2683.3 248.5 0.0 1)

(-3358.2 248.5 -10.0 1);;239
(-3692.0 248.5 -10.0 1)
(-3692.0 452.0 -10.0 1)
(-3692.0 452.0 0.0 1)
(-3692.0 248.5 0.0 1)
(-3358.2 248.5 0.0 1)

(-3692.0 553.0 -10.0 1);;245
(-3692.0 727.3 -10.0 1)
(-3711.6 727.3 -10.0 1)
(-3711.6 809.7 -10.0 1)
(-3799.7 809.7 -10.0 1)
(-3799.7 809.7 0.0 1)
(-3711.6 809.7 0.0 1)
(-3711.6 727.3 0.0 1)
(-3692.0 727.3 0.0 1)
```

```
(-3692.0 553.0 0.0 1)

(-3962.2 809.7 -10.0 1);;255
(-3962.2 363.9 -10.0 1)
(-3962.2 363.9 0.0 1)
(-3962.2 809.7 0.0 1)

(-3962.2 248.5 -10.0 1);;259
(-5264.7 248.5 -10.0 1)
(-5264.7 248.5 0.0 1)
(-3962.2 248.5 0.0 1)

(-5365.7 248.5 -10.0 1);;263
(-5680.7 248.5 -10.0 1)
(-5680.7 375.0 -10.0 1)
(-5819.2 375.0 -10.0 1)
(-5819.2 375.0 0.0 1)
(-5680.7 375.0 0.0 1)
(-5680.7 248.5 0.0 1)
(-5365.7 248.5 0.0 1)

(-5819.2 274.0 -10.0 1);;271
(-5819.2 248.5 -10.0 1)
(-6239.2 248.5 -10.0 1)
(-6239.2 248.5 0.0 1)
(-5819.2 248.5 0.0 1)
(-5819.2 274.0 0.0 1)

(-6239.2 0.0 -10.0 1);;277
(-5937.7 0.0 -10.0 1)
(-5937.7 0.0 0.0 1)
(-6239.2 0.0 0.0 1)

(-5836.7 0.0 -10.0 1);;281
(-5369.2 0.0 -10.0 1)
(-5369.2 0.0 0.0 1)
(-5836.7 0.0 0.0 1)

(-5268.2 0.0 -10.0 1);;285
(-4801.2 0.0 -10.0 1)
(-4801.2 0.0 0.0 1)
(-5268.2 0.0 0.0 1)
```

```
(-4711.7 0.0 -10.0 1);;289
(-4534.0 0.0 -10.0 1)
(-4534.0 -48.0 -10.0 1)
(-4447.9 -48.0 -10.0 1)
(-4447.9 0.0 -10.0 1)
(-4371.9 0.0 -10.0 1)
(-4371.9 0.0 0.0 1)
(-4447.9 0.0 0.0 1)
(-4447.9 -48.0 0.0 1)
(-4534.0 -48.0 0.0 1)
(-4534.0 0.0 0.0 1)
(-4711.7 0.0 0.0 1)

(-4282.4 0.0 -10.0 1);;301 +5
(-4232.4 0.0 -10.0 1);;-5
(-4232.4 0.0 0.0 1)
(-4282.4 0.0 0.0 1)

(-4041.4 0.0 -10.0 1);;305
(-4004.8 0.0 -10.0 1)
(-4004.8 0.0 0.0 1)
(-4041.4 0.0 0.0 1)

(-3914.3 0.0 -10.0 1);;309
(-3356.3 0.0 -10.0 1)
(-3356.3 0.0 0.0 1)
(-3914.3 0.0 0.0 1)

(-3255.3 0.0 -10.0 1);;313
(-3023.8 0.0 -10.0 1)
(-3023.8 0.0 0.0 1)
(-3255.3 0.0 0.0 1)

(-3023.8 -13.25 -10.0 1);;317
(-3114.0 -13.25 -10.0 1)
(-3114.0 -89.25 -10.0 1)
(-3131.4 -89.25 -10.0 1)
(-3131.4 -89.25 0.0 1)
(-3114.0 -89.25 0.0 1)
(-3114.0 -13.25 0.0 1)
(-3023.8 -13.25 0.0 1)
```

```
(-3131.4 -190.25 -10.0 1);;325
(-3131.4 -609.85 -10.0 1)
(-2576.0 -609.85  -10.0 1)
(-2576.0 -206.65 -10.0 1);;-5
(-2576.0 -206.65 0.0 1);;-5
(-2576.0 -609.85 0.0 1)
(-3131.4 -609.85 0.0 1)
(-3131.4 -190.25 0.0 1)

(-2565.0 -206.65 -10.0 1);;333
(-2565.0 -609.85 -10.0 1)
(-2014.0 -609.85 -10.0 1)
(-2014.0 -357.25 -10.0 1)
(-2014.0 -357.25 0.0 1)
(-2014.0 -609.85 0.0 1)
(-2565.0 -609.85 0.0 1)
(-2565.0 -206.65 0.0 1)

(-2014.0 -256.25 -10.0 1);;341
(-2014.0 -89.25 -10.0 1)
(-2032.0 -89.25 -10.0 1)
(-2032.0 -13.25 -10.0 1)
(-2032.0 -13.25 0.0 1)
(-2032.0 -89.25 0.0 1)
(-2014.0 -89.25 0.0 1)
(-2014.0 -256.25 0.0 1)

(-2027.0 0.0 -10.0 1);;349
(-1976.4 0.0 -10.0 1)
(-1976.4 0.0 0.0 1)
(-2027.0 0.0 0.0 1)

(-1875.4 0.0 -10.0 1);;353
(-1558.2 0.0 -10.0 1)
(-1558.2 0.0 0.0 1)
(-1875.4 0.0 0.0 1)

(-1457.2 0.0 -10.0 1);;357
(-1316.9 0.0 -10.0 1)
(-1316.9 0.0 0.0 1)
(-1457.2 0.0 0.0 1)
```

```
(-1215.9 0.0 -10.0 1);;361
(-839.4 0.0 -10.0 1)
(-839.4 0.0 0.0 1)
(-1215.9 0.0 0.0 1)

(-738.4 0.0 -10.0 1);;365
(-507.1 0.0 -10.0 1)
(-507.1 0.0 0.0 1)
(-738.4 0.0 0.0 1)

(-484.6 0.0 -10.0 1);;369
(-422.1 0.0 -10.0 1)
(-422.1 0.0 0.0 1)
(-484.6 0.0 0.0 1)

(-321.1 0.0 -10.0 1);;373
(-270.9 0.0 -10.0 1)
(-270.9 0.0 0.0 1)
(-321.1 0.0 0.0 1)

(-169.9 0.0 -10.0 1);;377
(0.0 0.0 -10.0 1)
(0.0 0.0 0.0 1)
(-169.9 0.0 0.0 1)

;;doors
(0.0 48.25 0.0 1);;381
(0.0 48.25 -212.5 1)
(0.0 200.75 -212.5 1)
(0.0 200.75 0.0 1)

(0.0 124.25 0.0 1);;385
(0.0 124.25 -212.5 1)

(-164.7 248.5 0.0 1);;387
(-164.7 248.5 -212.5 1)
(-245.7 248.5 -212.5 1)
(-245.7 248.5 0.0 1)
(-245.7 248.5 -212.5 1)
(-164.7 248.5 -212.5 1)
```

```
(-166.2 256.75 0.0 1);;393
(-244.2 256.75 0.0 1)
(-244.2 256.75 -212.5 1)
(-166.2 256.75 -212.5 1)

(-390.2 248.5 0.0 1);;397
(-390.2 248.5 -212.5 1)
(-491.2 248.5 -212.5 1)
(-491.2 248.5 0.0 1)
(-491.2 248.5 -212.5 1)
(-390.2 248.5 -212.5 1)

(-391.7 256.75 0.0 1);;403
(-489.7 256.75 0.0 1)
(-489.7 256.75 -212.5 1)
(-391.7 256.75 -212.5 1)

(-662.8 845.5 0.0 1);;407 -- big door
(-662.8 845.5 -212.5 1)
(-815.3 845.5 -212.5 1)
(-815.3 845.5 0.0 1)

(-739.05 845.5 0.0 1);;411
(-739.05 845.5 -212.5 1)

(-1190.5 248.5 0.0 1);;413
(-1190.5 248.5 -212.5 1)
(-1281.5 248.5 -212.5 1)
(-1281.5 248.5 0.0 1)
(-1281.5 248.5 -212.5 1)
(-1190.5 248.5 -212.5 1)

(-1192.0 256.75 0.0 1);;419
(-1280.0 256.75 0.0 1)
(-1280.0 256.75 -212.5 1)
(-1192.0 256.75 -212.5 1)

(-1592.9 248.5 0.0 1);;423
(-1592.9 248.5 -212.5 1)
(-1683.9 248.5 -212.5 1)
(-1683.9 248.5 0.0 1)
(-1683.9 248.5 -212.5 1)
```

```
(-1592.9 248.5 -212.5 1)

(-1594.4 256.75 0.0 1);;429
(-1682.4 256.75 0.0 1)
(-1682.4 256.75 -212.5 1)
(-1594.4 256.75 -212.5 1)

(-2159.3 248.5 0.0 1);;433
(-2159.3 248.5 -212.5 1)
(-2250.3 248.5 -212.5 1)
(-2250.3 248.5 0.0 1)
(-2250.3 248.5 -212.5 1)
(-2159.3 248.5 -212.5 1)

(-2160.8 256.75 0.0 1);;439
(-2248.8 256.75 0.0 1)
(-2248.8 256.75 -212.5 1)
(-2160.8 256.75 -212.5 1)

(-2587.3 248.5 0.0 1);;443
(-2587.3 248.5 -212.5 1)
(-2678.3 248.5 -212.5 1)
(-2678.3 248.5 0.0 1)
(-2678.3 248.5 -212.5 1)
(-2587.3 248.5 -212.5 1)

(-2588.8 256.75 0.0 1);;449
(-2676.8 256.75 0.0 1)
(-2676.8 256.75 -212.5 1)
(-2588.8 256.75 -212.5 1)

(-3692.0 457.0 0.0 1);;453 vertical
(-3692.0 457.0 -212.5 1)
(-3692.0 548.0 -212.5 1)
(-3692.0 548.0 0.0 1)
(-3692.0 548.0 -212.5 1)
(-3692.0 457.0 -212.5 1)

(-3683.75 458.5 0.0 1);;459
(-3683.75 546.5 0.0 1)
(-3683.75 546.5 -212.5 1)
(-3683.75 458.5 -212.5 1)
```

```
(-3804.7 809.7 0.0 1);;463 -- big door
(-3804.7 809.7 -212.5 1)
(-3957.2 809.7 -212.5 1)
(-3957.2 809.7 0.0 1)

(-3880.95 809.7 0.0 1);;467
(-3880.95 809.7 -212.5 1)

(-3962.2 358.9 0.0 1);;469 vertical
(-3962.2 358.9 -212.5 1)
(-3962.2 267.9 -212.5 1)
(-3962.2 267.9 0.0 1)
(-3962.2 267.9 -212.5 1)
(-3962.2 358.9 -212.5 1)

(-3970.45 357.4 0.0 1);;475
(-3970.45 269.4 0.0 1)
(-3970.45 269.4 -212.5 1)
(-3970.45 357.4 -212.5 1)

(-5269.7 248.5 0.0 1);;479
(-5269.7 248.5 -212.5 1)
(-5360.7 248.5 -212.5 1)
(-5360.7 248.5 0.0 1)
(-5360.7 248.5 -212.5 1)
(-5269.7 248.5 -212.5 1)

(-5271.2 256.75 0.0 1);;485
(-5359.2 256.75 0.0 1)
(-5359.2 256.75 -212.5 1)
(-5271.2 256.75 -212.5 1)

(-5819.2 370.0 0.0 1);;489 ;;auditorium
(-5819.2 370.0 -212.5 1)
(-5819.2 279.0 -212.5 1)
(-5819.2 279.0 0.0 1)

(-5932.7 0.0 0.0 1);;493
(-5932.7 0.0 -212.5 1)
(-5841.7 0.0 -212.5 1)
(-5841.7 0.0 0.0 1)
```

```
(-5841.7 0.0 -212.5 1)
(-5932.7 0.0 -212.5 1)

(-5931.2 -8.25 0.0 1);;499
(-5843.2 -8.25 0.0 1)
(-5843.2 -8.25 -212.5 1)
(-5931.2 -8.25 -212.5 1)

(-5364.2 0.0 0.0 1);;503
(-5364.2 0.0 -212.5 1)
(-5273.2 0.0 -212.5 1)
(-5273.2 0.0 0.0 1)
(-5273.2 0.0 -212.5 1)
(-5364.2 0.0 -212.5 1)

(-5362.7 -8.25 0.0 1);;509
(-5274.7 -8.25 0.0 1)
(-5274.7 -8.25 -212.5 1)
(-5362.7 -8.25 -212.5 1)

(-4796.2 0.0 0.0 1);;513
(-4796.2 0.0 -212.5 1)
(-4715.7 0.0 -212.5 1)
(-4715.7 0.0 0.0 1)
(-4715.7 0.0 -212.5 1)
(-4796.2 0.0 -212.5 1)

(-4794.7 -8.25 0.0 1);;519
(-4717.2 -8.25 0.0 1)
(-4717.2 -8.25 -212.5 1)
(-4794.7 -8.25 -212.5 1)

(-4367.9 0.0 0.0 1);;523
(-4367.9 0.0 -212.5 1)
(-4287.4 0.0 -212.5 1)
(-4287.4 0.0 0.0 1)
(-4287.4 0.0 -212.5 1)
(-4367.9 0.0 -212.5 1)

(-4366.4 -8.25 0.0 1);;529
(-4288.9 -8.25 0.0 1)
(-4288.9 -8.25 -212.5 1)
```

```
(-4366.4 -8.25 -212.5 1)

(-4228.4 0.0 0.0 1);;533
(-4228.4 0.0 -212.5 1)
(-4046.4 0.0 -212.5 1)
(-4046.4 0.0 0.0 1)
(-4046.4 0.0 -212.5 1)
(-4228.4 0.0 -212.5 1)

(-4226.9 -8.25 0.0 1);;539
(-4047.9 -8.25 0.0 1)
(-4047.9 -8.25 -212.5 1)
(-4226.9 -8.25 -212.5 1)

(-3999.8 0.0 0.0 1);;543
(-3999.8 0.0 -212.5 1)
(-3919.3 0.0 -212.5 1)
(-3919.3 0.0 0.0 1)
(-3919.3 0.0 -212.5 1)
(-3999.8 0.0 -212.5 1)

(-3998.3 -8.25 0.0 1);;549
(-3920.8 -8.25 0.0 1)
(-3920.8 -8.25 -212.5 1)
(-3998.3 -8.25 -212.5 1)

(-3351.3 0.0 0.0 1);;553
(-3351.3 0.0 -212.5 1)
(-3260.3 0.0 -212.5 1)
(-3260.3 0.0 0.0 1)
(-3260.3 0.0 -212.5 1)
(-3351.3 0.0 -212.5 1)

(-3349.8 -8.25 0.0 1);;559
(-3261.8 -8.25 0.0 1)
(-3261.8 -8.25 -212.5 1)
(-3349.8 -8.25 -212.5 1)

(-3131.4 -94.25 0.0 1);;563 room 242
(-3131.4 -94.25 -212.5 1)
(-3131.4 -185.25 -212.5 1)
(-3131.4 -185.25 0.0 1)
```

```
(-2014.0 -352.25 0.0 1);;567 room 242
(-2014.0 -352.25 -212.5 1)
(-2014.0 -261.25 -212.5 1)
(-2014.0 -261.25 0.0 1)

(-1971.4 0.0 0.0 1);;571
(-1971.4 0.0 -212.5 1)
(-1880.4 0.0 -212.5 1)
(-1880.4 0.0 0.0 1)
(-1880.4 0.0 -212.5 1)
(-1971.4 0.0 -212.5 1)

(-1969.9 -8.25 0.0 1);;577
(-1881.9 -8.25 0.0 1)
(-1881.9 -8.25 -212.5 1)
(-1969.9 -8.25 -212.5 1)

(-1553.2 0.0 0.0 1);;581
(-1553.2 0.0 -212.5 1)
(-1462.2 0.0 -212.5 1)
(-1462.2 0.0 0.0 1)
(-1462.2 0.0 -212.5 1)
(-1553.2 0.0 -212.5 1)

(-1551.7 -8.25 0.0 1);;587
(-1463.7 -8.25 0.0 1)
(-1463.7 -8.25 -212.5 1)
(-1551.7 -8.25 -212.5 1)

(-1311.9 0.0 0.0 1);;591
(-1311.9 0.0 -212.5 1)
(-1220.9 0.0 -212.5 1)
(-1220.9 0.0 0.0 1)
(-1220.9 0.0 -212.5 1)
(-1311.9 0.0 -212.5 1)

(-1310.4 -8.25 0.0 1);;597
(-1222.4 -8.25 0.0 1)
(-1222.4 -8.25 -212.5 1)
(-1310.4 -8.25 -212.5 1)
```

```
(-834.4 0.0 0.0 1);;601
(-834.4 0.0 -212.5 1)
(-743.4 0.0 -212.5 1)
(-743.4 0.0 0.0 1)
(-743.4 0.0 -212.5 1)
(-834.4 0.0 -212.5 1)

(-832.9 -8.25 0.0 1);;607
(-744.9 -8.25 0.0 1)
(-744.9 -8.25 -212.5 1)
(-832.9 -8.25 -212.5 1)

(-417.1 0.0 0.0 1);;611
(-417.1 0.0 -212.5 1)
(-326.1 0.0 -212.5 1)
(-326.1 0.0 0.0 1)
(-326.1 0.0 -212.5 1)
(-417.1 0.0 -212.5 1)

(-415.6 -8.25 0.0 1);;617
(-327.6 -8.25 0.0 1)
(-327.6 -8.25 -212.5 1)
(-415.6 -8.25 -212.5 1)

(-265.9 0.0 0.0 1);;621
(-265.9 0.0 -212.5 1)
(-174.9 0.0 -212.5 1)
(-174.9 0.0 0.0 1)
(-174.9 0.0 -212.5 1)
(-265.9 0.0 -212.5 1)

(-264.4 -8.25 0.0 1);;627
(-176.4 -8.25 0.0 1)
(-176.4 -8.25 -212.5 1)
(-264.4 -8.25 -212.5 1)

(-893.5 771.9 0.0 1);; 631 elev.
(-893.5 771.9 -212.5 1)
(-893.5 635.9 -212.5 1)
(-893.5 635.9 0.0 1)
(-893.5 635.9 -212.5 1)
(-893.5 771.9 -212.5 1)
```

```
(-956.5 771.9 0.0 1);;637
(-956.5 771.9 -212.5 1)
(-956.5 635.9 -212.5 1)
(-956.5 635.9 0.0 1)

(-893.5 477.4 0.0 1);; 641 elev.
(-893.5 477.4 -212.5 1)
(-893.5 341.7 -212.5 1)
(-893.5 341.7 0.0 1)
(-893.5 341.7 -212.5 1)
(-893.5 477.4 -212.5 1)

(-956.5 477.4 0.0 1);;647
(-956.5 477.4 -212.5 1)
(-956.5 341.7 -212.5 1)
(-956.5 341.7 0.0 1)

(-4137.4 -8.25 0.0 1);;651
(-4137.4 -8.25 -212.5 1)

;;end doors

;;ceiling
(-657.8 248.5 0.0 1) ;;653
(-657.8 248.5 -212.5 1)
(-893.5 248.5 -212.5 1)
(-893.5 248.5 0.0 1)
(-893.5 248.5 -212.5 1)
(-893.5 262.5 -212.5 1)
(-893.5 262.5 -286.5 1)
(-893.5 262.5 -212.5 1)
(-657.8 262.5 -212.5 1)
(-657.8 262.5 -286.5 1)
(-657.8 262.5 -212.5 1)
(-657.8 248.5 -212.5 1)

(-893.5 262.5 -286.5 1);;665
(-657.8 262.5 -286.5 1)
(-657.8 845.5 -286.5 1)
(-657.8 845.5 0.0 1)
(-657.8 845.5 -286.5 1)
```

```
(-893.5 845.5 -286.5 1)
(-893.5 845.5 0.0 1)
(-893.5 845.5 -286.5 1)

(-3692.0 248.5 0.0 1) ;;673
(-3692.0 248.5 -212.5 1)
(-3962.2 248.5 -212.5 1)
(-3962.2 248.5 0.0 1)
(-3962.2 248.5 -212.5 1)
(-3962.2 262.5 -212.5 1)
(-3962.2 262.5 -286.5 1)
(-3962.2 262.5 -212.5 1)
(-3692.0 262.5 -212.5 1)
(-3692.0 262.5 -286.5 1)
(-3692.0 262.5 -212.5 1)
(-3692.0 248.5 -212.5 1)

(-3962.2 262.5 -286.5 1);;685
(-3692.0 262.5 -286.5 1)
(-3692.0 727.3 -286.5 1)
(-3692.0 727.3 0.0 1)
(-3692.0 727.3 -286.5 1)
(-3711.6 727.3 -286.5 1)
(-3711.6 727.3 0.0 1)
(-3711.6 727.3 -286.5 1)
(-3711.6 809.7 -286.5 1)
(-3711.6 809.7 0.0 1)
(-3711.6 809.7 -286.5 1)
(-3962.2 809.7 -286.5 1)
(-3962.2 809.7 0.0 1)
(-3962.2 809.7 -286.5 1)


(-5680.7 248.5 0.0 1) ;;699
(-5680.7 248.5 -212.5 1)
(-5819.2 248.5 -212.5 1)
(-5819.2 248.5 0.0 1)
(-5819.2 248.5 -212.5 1)
(-5819.2 276.0 -212.5 1)
(-5819.2 276.0 -217.5 1)
(-5819.2 276.0 -212.5 1)
(-5680.7 276.0 -212.5 1)
```

```
(-5680.7 276.0 -217.5 1)
(-5680.7 276.0 -212.5 1)
(-5680.7 248.5 -212.5 1)

(-5819.2 276.0 -217.5 1);;711
(-5680.7 276.0 -217.5 1)
(-5680.7 375.5 -217.5 1)
(-5680.7 375.5 0.0 1)
(-5680.7 375.5 -217.5 1)
(-5819.2 375.5 -217.5 1)
(-5819.2 375.5 0.0 1)
(-5819.2 375.5 -217.5 1)


(-4534.0 0.0 0.0 1);;719
(-4534.0 0.0 -212.5 1)
(-4534.0 -48.0 -212.5 1)
(-4534.0 -48.0 0.0 1)

(-4447.9 0.0 0.0 1);;723
(-4447.9 0.0 -212.5 1)
(-4447.9 -48.0 -212.5 1)
(-4447.9 -48.0 0.0 1)

;;water fountain
(-4516.9 0.0 0.0 1);;727
(-4516.9 0.0 -94.0 1)
(-4516.9 -31.0 -94.0 1)
(-4516.9 -31.0 0.0 1)

(-4485.9 0.0 0.0 1);;731
(-4485.9 0.0 -94.0 1)
(-4485.9 -31.0 -94.0 1)
(-4485.9 -31.0 0.0 1)

;;box1
(-2729.2 -491.85 0.0 1);;735
(-2729.2 -491.85 -61.0 1)
(-2729.2 -427.85 -61.0 1)
(-2729.2 -427.85 0.0 1)

(-2973.2 -491.85 0.0 1);;739
```

```
(-2973.2 -491.85 -61.0 1)
(-2973.2 -427.85 -61.0 1)
(-2973.2 -427.85 0.0 1)

;;box2
(-2729.2 -276.85 0.0 1);;743
(-2729.2 -276.85 -61.0 1)
(-2729.2 -152.85 -61.0 1)
(-2729.2 -152.85 0.0 1)

(-2973.2 -276.85 0.0 1);;747
(-2973.2 -276.85 -61.0 1)
(-2973.2 -152.85 -61.0 1)
(-2973.2 -152.85 0.0 1)

;;elev1
(-956.5 818.9 0.0 1);;751
(-956.5 818.9 -233.0 1)
(-1096.68 818.9 -233.0 1)
(-1096.68 818.9 0.0 1)

(-956.5 588.9 0.0 1);;755
(-956.5 588.9 -233.0 1)
(-1096.68 588.9 -233.0 1)
(-1096.68 588.9 0.0 1)

;;elev2
(-956.5 524.7 0.0 1);;759
(-956.5 524.7 -233.0 1)
(-1096.68 524.7 -233.0 1)
(-1096.68 524.7 0.0 1)

(-956.5 294.7 0.0 1);;763
(-956.5 294.7 -233.0 1)
(-1096.68 294.7 -233.0 1)
(-1096.68 294.7 0.0 1)

;;mail
(-507.1 0.0 0.0 1);;767
(-507.1 0.0 -257 1)
(-484.6 0.0 -257 1)
(-484.6 0.0 0.0 1)
```

```
;;grid
(-3276.8 248.5 0.0 1);;771
(-3358.2 248.5 0.0 1)
(-3358.2 248.5 -45.5 1)
(-3276.8 248.5 -45.5 1)


;;island scrub board extern
(-2922.0 0.0 0.0 1);;775
(-2922.0 0.0 -10.0 1)
(-2128.0 0.0 -10.0 1)
(-2128.0 0.0 0.0 1)

(-2927.0 0.0 0.0 1);;779
(-2927.0 0.0 -212.5 1)
(-3018.8 0.0 -212.5 1)
(-3018.8 0.0 0.0 1)

(-2123.0 0.0 0.0 1);;783
(-2123.0 0.0 -212.5 1)
(-2032.0 0.0 -212.5 1)
(-2032.0 0.0 0.0 1)


;;242 ceiling
(-2545.0 -13.25 -370.0 1);;787
(-2545.0 -89.25 -370.0 1)
(-2596.0 -89.25 -370.0 1)
(-2596.0 -13.25 -370.0 1)
(-3114.0 -13.25 -370.0 1)
(-3114.0 -89.25 -370.0 1)
(-3131.4 -89.25 -370.0 1)
(-3131.4 -609.85 -370.0 1)
(-2014.0 -609.85 -370.0 1)
(-2014.0 -89.25 -370.0 1)
(-2032.0 -89.25 -370.0 1)
(-2032.0 -13.25 -370.0 1)


(-2545.0 -13.25 0.0 1);;799
(-2545.0 -89.25 0.0 1)
```

```
(-2596.0 -89.25 0.0 1)
(-2596.0 -13.25 0.0 1)
(-3114.0 -13.25 0.0 1)
(-3114.0 -89.25 0.0 1)
(-3131.4 -89.25 0.0 1)
(-3131.4 -609.85 0.0  1)
(-2014.0 -609.85 0.0 1)
(-2014.0 -89.25 0.0 1)
(-2032.0 -89.25 0.0 1)
(-2032.0 -13.25 0.0 1)

;;island scrub board intern
(-2128.0 -13.25 0.0 1);;811
(-2545.0 -13.25 0.0 1)
(-2545.0 -89.25 0.0 1)
(-2565.0 -89.25 0.0 1)
(-2565.0 -104.25 0.0 1)
(-2565.0 -104.25 -10.0 1)
(-2565.0 -89.25 -10.0 1)
(-2545.0 -89.25 -10.0 1)
(-2545.0 -13.25 -10.0 1)
(-2128.0 -13.25 -10.0 1)

(-2576.0 -104.25 0.0 1);;821
(-2576.0 -89.25 0.0 1)
(-2596.0 -89.25 0.0 1)
(-2596.0 -13.25 0.0 1)
(-2922.0 -13.25 0.0 1)
(-2922.0 -13.25 -10.0 1)
(-2596.0 -13.25 -10.0 1)
(-2596.0 -89.25 -10.0 1)
(-2576.0 -89.25 -10.0 1)
(-2576.0 -104.25 -10.0 1)

;;wall between 240 242
(-2565.0 -89.25 0.0 1);;831
(-2565.0 -89.25 -246.0 1)
(-2576.0 -89.25 -246.0 1)
(-2576.0 -89.25 0.0 1)

(-2565.0 -609.85 0.0 1);;835
(-2565.0 -609.85 -246.0 1)
```

```
      (-2576.0 -609.85 -246.0 1)
      (-2576.0 -609.85 0.0 1)


      ;;door between 240 242
      (-2565.0 -201.65 0.0 1);;839
      (-2576.0 -201.65 0.0 1)
      (-2576.0 -201.65 -212.5 1)
      (-2565.0 -201.65 -212.5 1)

      (-2565.0 -109.25 0.0 1);;843
      (-2576.0 -109.25 0.0 1)
      (-2576.0 -109.25 -212.5 1)
      (-2565.0 -109.25 -212.5 1)


      ;;island
      (-2123.0 -13.25 0.0 1);;847
      (-2545.0 -13.25 0.0 1)
      (-2545.0 -89.25 0.0 1)
      (-2565.0 -89.25 0.0 1)
      (-2565.0 -109.25 0.0 1)
      (-2576.0 -109.25 0.0 1)
      (-2576.0 -89.25 0.0 1)
      (-2596.0 -89.25 0.0 1)
      (-2596.0 -13.25 0.0 1)
      (-2927.0 -13.25 0.0 1)
      (-2927.0 0.0 0.0 1)
      (-2123.0 0.0 0.0 1)

)

(polygon-list *floor*)

'(   ;;floor
 ( 1 2 3 4 5 6 7 8 9 10
     11 12 13 14 15 16 17 18 19 20
     21 22 23 24 25 26 27 28 29 30
     31 32 33 34 35 36 37 38 39 40
     41 42 43 44 45 46 47 48 49 50
     51 52 53 54 55 56 57 58 59 60
     61 62 63 64 65 66 67 68 69 70
```

```
    71 72 73 74 75 76 77 78 79 80
    81 82 83 84 85 86 87 88 89 90
    91 92 93 94 95 96 97 98 99 100
    101 102 103 104 105 106 107 108 109 110
    111 112 113 114 115 116 117 118 119 120
    121 122 123 124 125 126 127 128 129 130
    131 132 133 134 135 136 137 138 139 140 141)
;;bebedouro
(142 143 144 145)
;;island
;;(146 147 148 149 150 151 152 153 154 155 156)
;;box1
(157 158 159 160)
;;box2
(161 162 163 164)
;;ceiling1
(165 166 167 168 169 170 171 172)
(1 165)(2 166)(69 167)(70 168)
;;ceiling2
(173 174 175 176 177 178 179 180 181 182)
;;ceiling3
(183 184 185 186)
(173 183)(174 184)(175 185)(176 186)

;;rodape
(187 188 189 190)
(191 192 193 194 195 196)
(197 198 199 200)
(201 202 203 204 205 206)
(207 208 209 210 211 212)
(213 214 215 216)
(217 218 219 220 221 222)
(223 224 225 226)
(227 228 229 230)
(231 232 233 234)
(235 236 237 238)
(239 240 241 242 243 244)
(245 246 247 248 249 250 251 252 253 254)
(255 256 257 258)
(259 260 261 262)
(263 264 265 266 267 268 269 270)
(271 272 273 274 275 276)
```

```
(277 278 279 280)
(281 282 283 284)
(285 286 287 288)
(289 290 291 292 293 294 295 296 297 298 299 300)
(301 302 303 304)
(305 306 307 308)
(309 310 311 312)
(313 314 315 316)
(317 318 319 320 321 322 323 324)
(325 326 327 328 329 330 331 332)
(333 334 335 336 337 338 339 340)
(341 342 343 344 345 346 347 348)
(349 350 351 352)
(353 354 355 356)
(357 358 359 360)
(361 362 363 364)
(365 366 367 368)
(369 370 371 372)
(373 374 375 376)
(377 378 379 380)

;;doors
(381 382 383 384)
(385 386)

(387 388 389 390 391 392)
(393 394 395 396)

(397 398 399 400 401 402)
(403 404 405 406)

(407 408 409 410)
(411 412)

(413 414 415 416 417 418)
(419 420 421 422)

(423 424 425 426 427 428)
(429 430 431 432)

(433 434 435 436 437 438)
(439 440 441 442)
```

```
(443 444 445 446 447 448)
(449 450 451 452)

(453 454 455 456 457 458)
(459 460 461 462)

(463 464 465 466)
(467 468)

(469 470 471 472 473 474)
(475 476 477 478)

(479 480 481 482 483 484)
(485 486 487 488)

(489 490 491 492)

(493 494 495 496 497 498)
(499 500 501 502)

(503 504 505 506 507 508)
(509 510 511 512)

(513 514 515 516 517 518)
(519 520 521 522)

(523 524 525 526 527 528)
(529 530 531 532)

(533 534 535 536 537 538)
(539 540 541 542)

(543 544 545 546 547 548)
(549 550 551 552)

(553 554 555 556 557 558)
(559 560 561 562)

(563 564 565 566)

(567 568 569 570)
```

```
(571 572 573 574 575 576)
(577 578 579 580)

(581 582 583 584 585 586)
(587 588 589 590)

(591 592 593 594 595 596)
(597 598 599 600)

(601 602 603 604 605 606)
(607 608 609 610)

(601 602 603 604 605 606)
(607 608 609 610)

(611 612 613 614 615 616)
(617 618 619 620)

(621 622 623 624 625 626)
(627 628 629 630)

(631 632 633 634 635 636)
(637 638 639 640)
(632 638) (633 639)

(641 642 643 644 645 646)
(647 648 649 650)
(642 648) (643 649)

(651 652)

;;ceiling
(653 654 655 656 657 658 659 660 661 662 663 664)

(665 666 667 668 669 670 671 672)

(673 674 675 676 677 678 679 680 681 682 683 684)

(685 686 687 688 689 690 691 692 693 694 695 696 697 698)

(699 700 701 702 703 704 705 706 707 708 709 710)
```

```
(711 712 713 714 715 716 717 718)

(719 720 721 722) (723 724 725 726) (720 724) (721 725)

(727 728 729 730) (731 732 733 734) (728 732) (729 733)

(735 736 737 738) (739 740 741 742) (736 740) (737 741)

(743 744 745 746) (747 748 749 750) (744 748) (745 749)

(751 752 753 754) (755 756 757 758) (752 756) (753 757)

(759 760 761 762) (763 764 765 766) (760 764) (761 765)

(767 768 769 770)

(771 772 773 774)

(775 776 777 778)

(779 780 781 782)

(783 784 785 786)

(787 788 789 790 791 792 793 794 795 796 797 798)
(787 799)(788 800)(789 801)(790 802)(791 803)(792 804)
(793 805)(794 806)(795 807)(796 808)(797 809)(798 810)

(811 812 813 814 815 816 817 818 819 820)

(821 822 823 824 825 826 827 828 829 830)

(831 832 833 834)
(835 836 837 838) (832 836)(833 837)

(839 840 841 842)
(843 844 845 846) (841 845)(842 846)

(847 848 849 850 851 852 853 854 855 856 857 858)

)))
```

APPENDIX H. C CODE FOR FINDING THE VISIBLE VERTICAL EDGES

```
/*****************************************************************************
  FILENAME: main.c
  PURPOSE: Main function for finding the visible vertical edges
                given a 2D map
  AUTHOR: Jader Gomes da Silva Filho
  DATE: 04/14/97
*****************************************************************************/
#include "path.h"
#include "display.h"
#include <stdio.h>
#include "utilities.h"


#define Tread = "r";
#define Twrite = "w";

int
main()
{

  Display_list *list;
  int i;
  long wid;
  Polygon *current_poly, *p1, *p2, *p3, *p4;
  Node_list *s_list;
  Vertex *goal, *start, *v1;
  Vertex_pair *vp;
  World *w;
  char *infilename1 = "floor2.dat";
  /*char *infilename1 = "positive.dat";*/
  char *infilename2 = "island.dat";
  char *infilename3 = "box1.dat";
  char *infilename4 = "box2.dat";
  char *outfilename = "output";
  char *outfilename1 = "outtest";
  char *type;
  CONFIGURATION q;
  int valid;
  FILE *infile;
  FILE *outfile;
  Stack *vis_stack;
  Point camera_pos;
```

```
  double dummy;

#ifdef IRIS
  /* debugging flag */
  foreground();
#endif

  q.point.x = 0.0;
  q.point.y = 0.0;
  q.theta = 0.0;
  q.kappa = 0.0;

  OpenFile(&outfile, outfilename,"w", "");
  /*outfile = fopen(outfilename,"w");*/

  /* make a world */
  w = create_world();

  /* make a polygon */
  p1 = create_polygon();

  /* add floor vertices */
  OpenFile(&infile, infilename1,"r", "");
  /*infile = fopen(infilename1,"r");*/

  while (!feof(infile)) {
    valid = fscanf(infile,"%lf %lf\n", &q.point.x, &q.point.y);
    add_vertex_to_polygon(q.point.x, q.point.y, p1);
    PrintFile(outfile, q);
    /*printf ("%lf %lf\n", q.point.x, q.point.y);
    fprintf (outfile,"%lf %lf\n", q.point.x, q.point.y);*/
  }
  PrintFile(outfile, p1->first_vertex->point);
  fclose(infile);
  fprintf (outfile,"\n");

  /* attach polygon to world */
  add_polygon_to_world(p1, w);

  /* make another polygon */
  p2 = create_polygon();
```

```
OpenFile(&infile, infilename2,"r", "");
/*infile = fopen(infilename2,"r");*/

while (!feof(infile)) {
  valid = fscanf(infile,"%lf %lf\n", &q.point.x, &q.point.y);
  add_vertex_to_polygon(q.point.x, q.point.y, p2);
  PrintFile(outfile, q);
  /*printf ("%lf %lf\n", q.point.x, q.point.y);
    fprintf (outfile,"%lf %lf\n", q.point.x, q.point.y);*/
}
PrintFile(outfile, p2->first_vertex->point);
fclose(infile);
fprintf (outfile,"\n");
add_polygon_to_world(p2, w);

/* create another polygon */
p3 = create_polygon();

OpenFile(&infile, infilename3,"r", "");
/*infile = fopen(infilename3,"r");*/

while (!feof(infile)) {
  valid = fscanf(infile,"%lf %lf\n", &q.point.x, &q.point.y);
  add_vertex_to_polygon(q.point.x, q.point.y, p3);
  PrintFile(outfile, q);
  /*printf ("%lf %lf\n", q.point.x, q.point.y);
  fprintf (outfile,"%lf %lf\n", q.point.x, q.point.y);*/
}
PrintFile(outfile, p3->first_vertex->point);
fclose(infile);
fprintf (outfile,"\n");
add_polygon_to_world(p3, w);

/* and another */
p4 = create_polygon();

OpenFile(&infile, infilename4,"r", "");
/*infile = fopen(infilename4,"r");*/

while (!feof(infile)) {
  valid = fscanf(infile,"%lf %lf\n", &q.point.x, &q.point.y);
  add_vertex_to_polygon(q.point.x, q.point.y, p4);
```

```
    PrintFile(outfile, q);
    /*printf ("%lf %lf\n", q.point.x, q.point.y);
    fprintf (outfile,"%lf %lf\n", q.point.x, q.point.y);*/
  }
  PrintFile(outfile, p4->first_vertex->point);
  fclose(infile);

  add_polygon_to_world(p4, w);

  /* create isolated vertex for start and goal */
  goal = create_isolated_vertex(5.0, 5.0);
  start = create_isolated_vertex(95.0, 95.0);

#ifdef IRIS
  /* initialize the display list */
  list = create_display_list();
  add_world_to_display_list(w, list);
  add_start_to_display_list(start, list);
  add_goal_to_display_list(goal, list);

  /* ok lets see it, init the screen and show the display list */
  wid = open_display_window();
#endif

  /* do the dijkstra search and show it progressively on the screen */
  s_list = dijkstra_search(start, goal, w, A_STAR, wid, list);

#ifdef IRIS
  /* display the final path */
  show_display_list(list, wid);
  display_best_path(s_list, wid);
  swapbuffers();

  while(1);
#endif

  fclose(outfile);
  Get_Cam_Pos(&camera_pos);
  vis_stack = Find_Visibility(p1,camera_pos);

  OpenFile(&outfile, outfilename1,"w", "");
  q.point = camera_pos;
```

```
  PrintFile(outfile, q);

  while (vis_stack != NULL)
    {
      pop(&vis_stack,&q.point,&dummy);
      PrintFile(outfile, q);
    }
  fclose(outfile);
  return 0;
}
```

```
/*****************************************************************************
  FILENAME: utilities.c
  PURPOSE: utility programs for spatial reasoning library
  CONTAINS: order()
tangent()
is_tangent()
common_tangent()
switch_mode()
segment_crossing_test()
linearize()
orientation()
normalize()
area()
fatal()
  AUTHOR: C.P. Lombardo
  DATE: 01/22/91
  CHANGES:      Jader Gomes da Silva Filho
  LAST UPDATE: 04/14/97
*****************************************************************************/

#include "spatial.h"
#include "utilities.h"
#include <stdio.h>



/*****************************************************************************
  FUNCTION:   order(p1, p2, p3)
  PARAMETERS: Point p1 the first point
              Point p2 the second point
      Point p3 the third point
  PURPOSE:    determine the order (i.e. CW or CCW) of three points
  RETURNS:    int  CW if clockwise
                   CCW if counterclockwise
  CALLED BY:  tangent()
              is_tangent()
      common_tangent()
      segment_crossing_test()
      vertex_pair_poly_crossing_test() <path.c>
      segment_splitting_test() <path.c>
      ANYBODY
```

```
  CALLS:      NONE
  COMMENTS:   can also be done with atan2() call
*****************************************************************************/

int order(p1, p2, p3)
Point p1;
Point p2;
Point p3;
{
  double area;

  /* calculate the area of the triangle */
  area = 0.5 * ((p2.x - p1.x) * (p3.y - p1.y) -
(p3.x - p1.x) * (p2.y - p1.y));

  if (area > 0.0)
    return(CCW);

  else if (area < 0.0)
    return(CW);

  else
    return(0);
}
/*****************************************************************************
  FUNCTION:   tangent(v, poly, mode)
  PARAMETERS: Vertex *v a vertex (i.e. point) outside of polygon poly
              Polygon *poly the polygon
      int mode the mode - either CCW or CW
  PURPOSE:    determine tangents from a point to a polygon
  RETURNS:    Vertex *
  CALLED BY:  init_node_costs() <path.c>
              update_node_costs() <path.c>
      ANYBODY
  CALLS:      is_tangent()
              order()
  COMMENTS:   currently convex polygons only
              O(n) search for the right vertex
*****************************************************************************/

Vertex *tangent(v, poly, mode)
Vertex *v;
```

```
Polygon *poly;
int mode;
{
  Vertex
    *current_vertex, /* the current vertex */
    *next_vertex, /* the next vertex after current_vertex */
    *previous_vertex;         /* the vertex before current_vertex */

  /* get the first vertex */
  current_vertex = poly->first_vertex;

  while(TRUE) {

    /* is this the right vertex */
    if(is_tangent(v->point, current_vertex, mode))
       break;

    /* now based on the mode move the proper way around the poly
     * go right for CCW  go left for CW
     */
    if(mode == CCW) {
      if(order(v->point, current_vertex->point,
                         current_vertex->next->point) >= 0)
current_vertex = current_vertex->previous;
      else
current_vertex = current_vertex->next;
    }

    else {
      if(order(v->point, current_vertex->point,
                         current_vertex->next->point) >= 0)
current_vertex = current_vertex->next;
      else
current_vertex = current_vertex->previous;
    }
  }

  return(current_vertex);
}
/*****************************************************************************
  FUNCTION:   is_tangent(pt, v, mode)
  PARAMETERS: Point pt the point away from the polygon
```

```
              Vertex *v the vertex on the polygon to test
        int mode the mode of the tangent sought - CCW or CW
  PURPOSE:    determine if pt and v make a tangent of the right mode
  RETURNS:    int :  1 - yes it is the right tangent
                     0 - not it is not the right tangent
  CALLED BY:  tangent()
              common_tangent()
        ANYBODY
  CALLS:      order()
  COMMENTS:   currently convex polygons only
*************************************************************************/

int is_tangent(pt, v, mode)
Point pt;
Vertex *v;
int mode;
{
  switch (mode) {

  case CCW: /* the + tangent */

    if(order(pt, v->point, v->next->point) >= 0 &&
       order(pt, v->point, v->previous->point) >= 0)
      return(1);

    break;

  case CW: /* the - tangent */

    if(order(pt, v->point, v->next->point) <= 0 &&
       order(pt, v->point, v->previous->point) <= 0)
      return(1);

  }

  /* other wise it is not the right tangent */
  return(0);
}
/*************************************************************************
  FUNCTION:   common_tangent(p1, p2, mode1, mode2)
  PARAMETERS: Polygon *p1 the first polygon
              Polygon *p2 the seconde polygon
```

```
      int mode1 the mode of the tangent for p1 - CCW or CW
      int mode2 the mode for p2 - either CCW or CW
  PURPOSE:    determine common tangents for two polygons
  RETURNS:    Vertex_pair *
  CALLED BY:  update_node_costs() <path.c>
              ANYBODY
  CALLS:      is_tangent()
              order()
      switch_mode()
  COMMENTS:   currently convex polygons only
*************************************************************************/

Vertex_pair *common_tangent(p1, p2, mode1, mode2)
Polygon *p1;
Polygon *p2;
int mode1;
int mode2;
{
  int
    p1_flag = 0, /* flag denoting a current tangent with p1 */
    p2_flag = 0; /* same for p2 */
  Vertex_pair
    *tangent_line; /* the resultant tangent line */
  Vertex
    *v1, /* the current vertex for p1 */
    *v2; /* the current vertex for p2 */

  /* get the first vertices */
  v1 = p1->first_vertex;
  v2 = p2->first_vertex;

  while(p1_flag == 0 || p2_flag == 0) {

    /* check the poly1 tangent using the point from poly2 */
    if(is_tangent(v1->point, v2, mode2))
      p2_flag = 1;

    else {
      p2_flag = 0;

      /* move v2 accordingly */
      if(mode2 == CCW) {
```

```
if(order(v1->point, v2->point, v2->next->point) >= 0)
  v2 = v2->previous;
else
  v2 = v2->next;
      }

      else {
if(order(v1->point, v2->point, v2->next->point) >= 0)
  v2 = v2->next;
else
  v2 = v2->previous;
      }
    }

    /* now check poly2 - note that the mode is switched: CW to CCW, CCW to CW */
    if(is_tangent(v2->point, v1, switch_mode(mode1)))
      p1_flag = 1;

    else {
      p1_flag = 0;

      /* move v1 accordingly */
      if(switch_mode(mode1) == CCW) {
if(order(v2->point, v1->point, v1->next->point) >= 0)
  v1 = v1->previous;
else
  v1 = v1->next;
      }

      else {
if(order(v2->point, v1->point, v1->next->point) >= 0)
  v1 = v1->next;
else
  v1 = v1->previous;
      }
    }
  }

  /* allocate space for the vertex pair */
  if((tangent_line = (Vertex_pair *)malloc(sizeof(Vertex_pair))) == NULL) {

    fatal("common_tangent: malloc\n");
```

```
    exit(FAILURE);
  }

  tangent_line->v1 = v1;
  tangent_line->v2 = v2;

  return(tangent_line);
}
/*****************************************************************************
  FUNCTION:   switch_mode(mode)
  PARAMETERS: int mode either CCW or CW
  PURPOSE:    return the opposite mode
  RETURNS:    int
  CALLED BY:  common_tangent()
              ANYBODY
  CALLS:      NONE
*****************************************************************************/
int switch_mode(mode)
int mode;
{
  if(mode == CCW)
    return(CW);
  else
    return(CCW);
}
/*****************************************************************************
  FUNCTION:   segment_crossing_test(l1, l2)
  PARAMETERS: Line_segment *l1 the first line segment
              Line_segment *l2 the second line segment
  PURPOSE:    determine if two line segments cross, touch or do not cross or
              touch
  RETURNS:    int: 1 - the line segments cross
                   0 - the line segments touch only but do not cross
       -1 - the line segments do not touch at all
  CALLED BY:  vertex_pair_poly_crossing_test() <path.c>
              ANYBODY
  CALLS:      order()
              linearize()
  COMMENTS:
*****************************************************************************/
int segment_crossing_test(l1, l2)
Line_segment *l1;
```

```
Line_segment *l2;
{
  double f1, f2, f3, f4; /* linearization values for the 4 points */
  int o1, o2, o3, o4; /* temp variables for keeping the orders */

  /* set up order calculations */
  o1 = order(l1->p1, l1->p2, l2->p1);
  o2 = order(l1->p1, l1->p2, l2->p2);
  o3 = order(l2->p1, l2->p2, l1->p1);
  o4 = order(l2->p1, l2->p2, l1->p2);

  /* test for crossing first - only one case */
  if(o1 != 0 && o2 != 0 && o1 != o2 &&
     o3 != 0 && o4 != 0 && o3 != o4)
    return(1);

  /* now test for touching  - three cases here one more below */
  if(((o1 == 0 || o2 == 0) && (o3 != 0 && o4 != 0 && o3 != o4)) ||
     ((o3 == 0 || o4 == 0) && (o1 != 0 && o2 != 0 && o1 != o2)) ||
     (((o1 == 0 || o2 == 0) && o1 != o2) && ((o3 == 0 || o4 == 0) && o3 != o4)))
    return(0);

  /* now test for colinear case */
  if(o1 == 0 && o2 == 0 && o3 == 0 && o4 == 0) {

    /* linearize on the first segment */
    f1 = linearize(l1->p1, l1);
    f2 = linearize(l1->p2, l1);
    f3 = linearize(l2->p1, l1);
    f4 = linearize(l2->p2, l1);

    /* case: non overlapping colinear segments */
    if((f3 < 0.0 && f4 < 0.0) || (f3 > f2 && f4 > f2))
      return(-1);

    /* case: overlapping segments */
    if((f3 <= f2 && f3 >= f1) || (f4 <= f2 && f4 >= f1))
      return(0);
  }

  /* only thing left is no touchy no crossy */
  return(-1);
```

```
}
/*****************************************************************************
  FUNCTION:   linearize(pt, line)
  PARAMETERS: Point pt point to check the linearized position
   on line segment line
      Line_segment *line the line segment to use as reference
  PURPOSE:    determine the linearize position of pt on line.  0 if pt
              corresponds to the first point in line, and some positive
      constant if pt corresponds to the 2nd point in the segment
  RETURNS:    double
  CALLED BY:  segment_crossing_test()
  CALLS:      NONE
  COMMENTS:   here the return value is given by:

              f(x, y) = (x - x1)(x2 - x1) + (y - y1)(y2 - y1)
*****************************************************************************/

double linearize(pt, line)
Point pt;
Line_segment *line;
{
  double
    f1, /* the (x - x1)(x2 - x1) part */
    f2; /* the (y - y1)(y2 - y1) part */

  f1 = (pt.x - line->p1.x) * (line->p2.x - line->p1.x);
  f2 = (pt.y - line->p1.y) * (line->p2.y - line->p1.y);

  return(f1 + f2);
}
/*****************************************************************************
  FUNCTION:   orientation(p1, p2)
  PARAMETERS: Point *p1 the first point of the line segment
              Point *p2 the second point of the line segment
  PURPOSE:    determine the angle of the line segment drawn from p1 to p2 with
              respect to the x axis. This angle by convention will be between
      -pi and pi.
  RETURNS:    double: the angle in radians
  CALLED BY:  ANYBODY
  CALLS:      NONE
*****************************************************************************/
double orientation(p1, p2)
```

```
Point *p1;
Point *p2;
{
  return(atan2(p2->y - p1->y, p2->x - p1->x));
}
/*****************************************************************************
  FUNCTION:   normalize(angle)
  PARAMETERS: double angle the angle to normalize
  PURPOSE:    normalize the input angle to the range -pi to pi.
  RETURNS:    double: the normalized angle in radians
  CALLED BY:  ANYBODY
  CALLS:      NONE
*****************************************************************************/
double normalize(angle)
double angle;
{
  /* test for greater than PI */
  while(angle > PI)
    angle -= 2 * PI;

  while(angle < -PI)
    angle += 2 * PI;

  return(angle);
}
/*****************************************************************************
  FUNCTION:   area(v0, v1, v2)
  PARAMETERS: Vertex *v0, *v1, *v2 the triangle vertices
  PURPOSE:    calculate the area of the triangle defined by the 3 line
              segments: v0 - v1, v1 - v2, and v2 - v0
  RETURNS:    double
  CALLED BY:  update_node_costs() <path.c>
              get_new_area() <path.c>
      ANYBODY
  CALLS:      NONE
  COMMENTS:   area given by A = 0.5[(x1y2-y1x2)-(x0y2-y0x2)+(x0y1-y0x1)]
              this is similar to order()
*****************************************************************************/

double area(v0, v1, v2)
Vertex *v0;
Vertex *v1;
```

```
Vertex *v2;
{
  double a1, a2, a3; /* the three parts of the area */

  a1 = (v1->point.x * v2->point.y) - (v1->point.y * v2->point.x);
  a2 = (v0->point.x * v2->point.y) - (v0->point.y * v2->point.x);
  a3 = (v0->point.x * v1->point.y) - (v0->point.y * v1->point.x);

  return(0.5 * (a1 - a2 + a3));
}
/*****************************************************************************
  FUNCTION:   fatal()
  PARAMETERS: char *message
  PURPOSE:    on fatal error print message then print system message then exit
  CALLED BY:  create_world() <world.c>
      create_polygon() <world.c>
      add_vertex_to_polygon() <world.c>
      create_isolated_vertex() <world.c>
      create_vertex_pair() <world.c>
      create_line_segment() <world.c>
      create_node_list() <path.c>
      create_start_or_goal_node() <path.c>
      add_node_to_node_list() <path.c>
      add_world_to_node_list() <path.c>
      create_display_list() <display.c>
      add_vertex_pair_to_display_list() <display.c>
      ANYBODY
  CALLS:      NONE
*****************************************************************************/

void fatal(message)
char *message;
{
  fprintf(stderr, "Fatal error occured: ");
  perror(message);
  exit(FAILURE);
}



      /*****************************************************************/
      /*                    FUNCTION: OpenFile()                       */
```

```
      /*                    --------------------                          */
      /* Purpose:  This function opens the output file.                   */
      /********************************************************************/
void OpenFile(file0, name, type, title)
FILE **file0;
char *name;
char *type;
char *title;

{
  *file0 = fopen(name,type);
  return;
}


      /********************************************************************/
      /*                    FUNCTION: PrintFile()                         */
      /*                    ---------------------                         */
      /* Purpose:  This function print the result to the file.            */
      /********************************************************************/

void PrintFile(file1, q)
FILE *file1;
CONFIGURATION q;
{
  fprintf (file1,"%f %f\n", q.point.x, q.point.y);
  return;
}


/*************************************************************************/
void push(temp_stack,pos,angle)
Point pos;
double angle;
Stack **temp_stack;
{
  Stack *new_stack_node;
  if((new_stack_node = (Stack *)malloc(sizeof(Stack))) == NULL) {
    fatal("push: malloc\n");
    exit(FAILURE);
  }
```

```
  new_stack_node->point = pos;
  new_stack_node->angle = angle;

  if (*temp_stack == NULL)
    {
printf("including first\n");
      new_stack_node->previous = new_stack_node;
      new_stack_node->next = new_stack_node;
    }
  else
    {
      new_stack_node->previous = (*temp_stack)->previous;
      (*temp_stack)->previous->next = new_stack_node;
      (*temp_stack)->previous = new_stack_node;
      new_stack_node->next = *temp_stack;
    }
printf ("Pushing %lf %lf\n", pos.x,pos.y);
  *temp_stack = new_stack_node;

  return;
}

/*************************************************************************/
void pop(temp_stack,temp_point,temp_angle)
Stack **temp_stack;
Point *temp_point;
double *temp_angle;

{
  Stack *temp_stk;

  temp_stk = NULL;

  if (*temp_stack == NULL) {return;}

  if ((*temp_stack)->previous != (*temp_stack)->previous->next)
    {
      temp_stk = (*temp_stack)->next;
      (*temp_stack)->next->previous = (*temp_stack)->previous;
      (*temp_stack)->previous->next = (*temp_stack)->next;

  *temp_angle = (*temp_stack)->angle;
```

```
  *temp_point = (*temp_stack)->point;
      printf ("Popping %lf %lf\n", temp_point->x,temp_point->y);
  free(*temp_stack);
  *temp_stack = temp_stk;

    }
  else {
    *temp_angle = (*temp_stack)->angle;
    *temp_point = (*temp_stack)->point;
    printf ("Popping LASTTTT %lf %lf\n", temp_point->x,temp_point->y);
    free(*temp_stack);
    *temp_stack = temp_stk;}
  return;
}

/*************************************************************************/
Stack *Find_Visibility(p1,pos)
Polygon *p1;
Point pos;


{
  Vertex *first_vertex,*current_vertex;
  double current_angle,last_angle,diff_angle,curr_dist,prev_dist,ang_max_left,ang
  Stack *stack_pointer;
  Point prev1,prev2, pop_point;
  int type, count, repeat, type_curve_left, type_curve_right, second_pass, type_c


  first_vertex = current_vertex = p1->first_vertex;
  stack_pointer = NULL;
  count = 0;
  repeat = FALSE;
  type = 0;
  second_pass = FALSE;

  type_curve_left = type_curve_right = type_curve_down = FALSE;

  last_angle = atan2(current_vertex->point.y-pos.y,
     current_vertex->point.x-pos.x);
  do
    {
```

```
      current_angle = atan2(current_vertex->point.y-pos.y,
    current_vertex->point.x-pos.x);
      diff_angle = normalize(current_angle - last_angle);

      if (diff_angle < 0.0 || stack_pointer == NULL)

{

  push(&stack_pointer,current_vertex->point,current_angle);
  count++;

  last_angle = current_angle;

}
      else
{
  if (stack_pointer == NULL)
    {
      printf("I'm leaving\n");
      return stack_pointer;
    }
  else
    {
      prev1 = stack_pointer->point;
      prev2 = stack_pointer->next->point;

      type1 = order(prev2,prev1,current_vertex->point);



      type = order(current_vertex->previous->point,
   current_vertex->point,current_vertex->next->point);

      prev_dist = pow(prev1.y-pos.y,2.0) + pow(prev1.x-pos.x,2.0);
      curr_dist = pow(current_vertex->point.y-pos.y,2.0) +
pow(current_vertex->point.x-pos.x,2.0);

      if (curr_dist <= prev_dist && type == CCW )
{
  printf ("Eliminating CW %lf %lf %lf %lf\n",prev1.x,prev1.y,
  curr_dist,prev_dist);
```

```
  pop(&stack_pointer,&pop_point,&last_angle);
  printf ("Popped point %lf %lf\n", pop_point.x,
  pop_point.y);
  if (stack_pointer != NULL)
    {
      last_angle = stack_pointer->angle;
    }
  count--;
  repeat = TRUE;
}
      else
{

  printf ("Eliminating CCW %lf %lf\n",
  current_vertex->point.x,
  current_vertex->point.y);
}
    }
}
      if (repeat == FALSE)
current_vertex = current_vertex->next;
      else
repeat = FALSE;
      if ((current_vertex == first_vertex)&&(second_pass == FALSE))
{
  second_pass  = TRUE;
  first_vertex = current_vertex->next;
  current_vertex->point = stack_pointer->previous->point;
  printf ("last %lf %lf %lf %lf %lf\n", current_vertex->point.x,
  current_vertex->point.y,current_angle,last_angle,diff_angle);

}

    } while(current_vertex != first_vertex);

  return stack_pointer;
}

/*************************************************************************/
void Get_Cam_Pos(Point *pos)

{
```

```
    printf("\nEnter Camera's X position :");scanf("%lf",&pos->x);
    printf("\nEnter Camera's Y position :");scanf("%lf",&pos->y);
    return;
}
```

APPENDIX I. C CODE FOR CONTROLLING "YAMABICO"

```
/*****************************************************************************
  Function   : user()
  Purpose    :
  Parameters: void
  Returns    : void
  Comments   : Kanayama and Jader Filho 04/25/97
*****************************************************************************/
#include "user.h"
#include "seqcmd.h"
#include "math.h"

void user10();
void user11();
void user12();
void user13();
void user14();
void user19();
void user20();
void user21();
void user22();
void user23();
void user24();
void user25();
void user26();
void user30();
void user100();
void user101();


int OnTrack(CONFIGURATION path)
{
  double lambda = 20.0;
  CONFIGURATION robot1;

  double kk;
  double k, kk2;
  LINE pathElement;
  double sigma = sigmaValue();

  pathElement.config = path;
  kk = path.Kappa ;
  kk2 = kk * kk;
```

```
  k = 1.0/sigma ;
  pathElement.a = 3.0*k ;
  pathElement.b = 3.0*k*k - kk2 ;
  pathElement.c = k*k*k - 3.0*k*kk2 ;

  robot1 = getRobotConfig();
  lambda = fabs(pathElement.a * robot1.Kappa)
    + fabs(pathElement.b *
   norm(robot1.Theta - pathElement.config.Theta))
    + fabs(pathElement.c *
   (-(robot1.Posit.X - pathElement.config.Posit.X)
    * sin(pathElement.config.Theta)
    + (robot1.Posit.Y - pathElement.config.Posit.Y)
    * cos(pathElement.config.Theta)));
  /*if((int)lambda%2 == 0)
  printf("\n%f",lambda);*/
  if (lambda > 0.0)
    return FALSE;
  else{
    /*    printf("\n%f",lambda);*/
    return TRUE;}
}


void user()
{
  int selection;

  printf("\n Continuous Curvature Motion Control: Version 4");

  printf("\n Enter 10 for Square");                  /* for lines    */
  printf("\n Enter 11 for Star");
  printf("\n Enter 12 for Neg Star");
  printf("\n Enter 13 for Maze");
  printf("\n Enter 19 for Polygon Tracking");

  printf("\n Enter 20 for CCW circle 100cm.");       /* for circles  */
  printf("\n Enter 21 for Tracking a circle form outside");
  printf("\n Enter 22 for Two circles");
  printf("\n Enter 23 for Circle Train");
  printf("\n Enter 24 for Slalom");
  printf("\n Enter 25 for Obstacle avoidance #1");
```

```
printf("\n Enter 26 for Obstacle avoidance #2");

printf("\n Enter 30 for Track_Stop");              /* others       */
printf("\n Enter 100 for Sonar Range Displaying"); /* others       */
printf("\n Enter 101 for Random Walk by Sonar");   /* others       */
printf("\n\n The choice is : ");

selection = GetInt();
switch (selection)
  {
  case 10:
    user10();
    break;
  case 11:
    user11();
    break;
  case 12:
    user12();
    break;
  case 13:
    user13();
    break;
  case 19:
    user19();
    break;
  case 20:
    user20();
    break;
  case 21:
    user21();
    break;
  case 22:
    user22();
    break;
  case 23:
    user23();
    break;
  case 24:
    user24();
    break;
  case 25:
    user25();
```

```
      break;
    case 26:
      user26();
      break;
    case 30:
      user30();
      break;
    case 100:
      user100();
      break;
    case 101:
      user101();
      break;
    default:
      break;
    }
}

/*****************************************************************************
  Function   : user101()
  Purpose    : Making random walk behavior, which turns left 135 degrees
  Parameters: void
  Returns    : void
  Comments   :
*****************************************************************************/
void user101()
{
  int dist, distL, distR;
  CONFIGURATION  path, left, right, current;
  double sigma = 6.0;

  path = defineConfig(0.0, 0.0, 0.0, 0.0);
  left = defineConfig(80.0, 0.0,+1.5*HPI, 0.0);
  right =defineConfig(80.0, 0.0,-1.5*HPI, 0.0);
  EnableSonar(S000);
  EnableSonar(S030);
  EnableSonar(S330);
  setLinVelImm(20.0);
  setSigmaImm(sigma);
  setRobotConfigImm(path);
  track(path);
```

```
  while (1)
    {
      waitMS(200);
      dist = Sonar(S000);
      distR = Sonar(S030);
      distL = Sonar(S330);
      if (dist < 150 && dist > 1)
{
  printf("\n distance= %d ", dist);
  DisableSonarInterrupts();
  current = getRobotConfig();
  current.Kappa = 0.0;
  if (distL > distR)
    path = compose(&current, &left);
  else
    path = compose(&current, &right);
  track(path);
  printf("\n turn %4.2f", path.Theta*r2d);
  waitSec(8);
  /*while(OnTrack(path) == FALSE);*/
  EnableSonarInterrupts();
  /*   dist = Sonar(S000);
  waitMS(60);*/
}else
  if (distL < 150 && distL > 1)
    {
      printf("\n distanceL= %d ", distL);
      DisableSonarInterrupts();
      current = getRobotConfig();
      current.Kappa = 0.0;
      path = compose(&current, &right);
      track(path);
      printf("\n turn %4.2f", path.Theta*r2d);
      waitSec(8);
      /* while(OnTrack(path) == FALSE);*/
      EnableSonarInterrupts();
      /*       distL = Sonar(S330);
      waitMS(60);*/
    }else
      if (distR < 150 && distR > 1)
{
  printf("\n distanceR= %d ", distR);
```

```
  DisableSonarInterrupts();
  current = getRobotConfig();
  current.Kappa = 0.0;
  path = compose(&current, &left);
  track(path);
  printf("\n turn %4.2f", path.Theta*r2d);
  waitSec(8);
  /*while(OnTrack(path) == FALSE);*/
  EnableSonarInterrupts();
  /*   distR = Sonar(S030);
  waitMS(60);*/
}
    }
  DisableSonar(S000);
  DisableSonar(S030);
  DisableSonar(S330);
  return;
}

void
user10(void)        /*  Square  */
{
  CONFIGURATION init, line, turn;
  double size = 100, sigma = 10;
  int i;
  printf("\nInput desired size (cm): ");
  size = GetReal();
  printf("\nInput desired smoothness (cm): ");
  sigma = GetReal();
  /*  setSigmaImm(size);
  setSigmaImm(0.05*size);*/
  setSigmaImm(sigma);
  init = defineConfig(-0.5*size, 0.0, 0.0, 0.0);
  line = defineConfig( 0.0, 0.0, 0.0, 0.0);
  turn = defineConfig( size, 0.0, HPI, 0.0);
  setRobotConfigImm(init);
  for (i = 0; i <= 4; i++)
    {
      track(line);
      line = compose(&line, &turn);
    }
  line = defineConfig( 1.5*size, 0.0, 0.0, 0.0);
```

```
  trackS(line);
  return;
}

void
user11(void)
{

   CONFIGURATION init, line0, line1;
   int i;
   double sigma, linelength;

   printf("\nInput desired smoothness (cm), eg. 7: ");
   /* A typical value for sigma is 7.0 */
   sigma = GetReal();

   setSigmaImm(sigma);

   printf("\nInput desired length (cm), eg. 300: ");
   /* A typical value for linelength is 300.0 */
   linelength = GetReal();

   init = defineConfig(linelength/2, 0.0, 0.0, 0.0);

   line0 = defineConfig(0.0, 0.0, 0.0, 0.0);
   line1 = defineConfig(linelength, 0.0, 4.0*PI/5.0, 0.0);

   setRobotConfigImm(init);

   for (i=0; i <= 4; i++)
     {
       track(line0);
       printf("\n line0.x = %f, line0.y = %f, line0.t = %f, line0.k = %f", line0.
       line0 = compose(&line0, &line1);
     }
   trackS(init);
   return;
}

void
user12(void)/* Neg_Star*/
{
```

```
    CONFIGURATION init, line0, line1;
    int i;
    double sigma, linelength;

    printf("\nInput desired smoothness (cm), eg. 7: ");
    /* A typical value for sigma is 7.0 */
    sigma = GetReal();

    setSigmaImm(sigma);

    printf("\nInput desired length (cm), eg. 300: ");
    /* A typical value for linelength is 300.0 */
    linelength = GetReal();

    init = defineConfig(linelength/2, 0.0, 0.0, 0.0);

    line0 = defineConfig(0.0, 0.0, 0.0, 0.0);
    line1 = defineConfig(linelength, 0.0, -4.0*PI/5.0, 0.0);

    setRobotConfigImm(init);

    for (i=0; i <= 4; i++)
      {
        track(line0);
        printf("\n line0.x = %f, line0.y = %f, line0.t = %f, line0.k = %f", line0.P(
        line0 = compose(&line0, &line1);
      }
    trackS(init);
    return;
}


void
user13(void)   /*  Maze   */
{
    CONFIGURATION init, line0, line1, line2, line3, line4, line5, line6, line7;
    double sigma;

    printf("\nInput desired smoothness (cm): ");
    sigma = GetReal();
    setSigmaImm(sigma);
```

```
    init = defineConfig(0.0, -50.0, 0.0, 0.0);
    line0 = defineConfig(0.0, -50.0, 0.0, 0.0);
    line1 = defineConfig(205.0, -50.0, HPI, 0.0);
    line2 = defineConfig(205.0, 306.0, 0.0, 0.0);
    line3 = defineConfig(399.0, 306.0, -HPI, 0.0);
    line4 = defineConfig(399.0, -50.0, PI, 0.0);
    line5 = defineConfig(205.0, 306.0, PI, 0.0);
    line6 = defineConfig(-50.0, 306.0, -HPI, 0.0);
    line7 = defineConfig( 50.0, -50.0, 0.0, 0.0);

    setRobotConfigImm(init);

    track(line0);
    track(line1);
    track(line2);
    track(line3);
    track(line4);
    track(line1);
    track(line5);
    track(line6);
    trackS(line7);
return;
}

void
user19(void)  /* polygon tracking   */
{
  CONFIGURATION init, line0, line1, line2;
  int i;
  double sigma;

  printf("\nInput desired smoothness (cm): ");
  sigma = GetReal();
  setSigmaImm(sigma);

  init = line0 = defineConfig(0.0, 0.0, 0.0, 0.0);
  line1 = defineConfig(200.0, 0.0, HPI, 0.0);
  line2 = defineConfig(200.0, 0.0, 0.0, 0.0);
  setRobotConfigImm(init);

  for (i=0; i <= 3; i++)
```

```
    {
      track(line0);
      line0 = compose(&line0, &line1);
    }
  trackS(line2);
  return;
}

void
user20(void)   /* CCW circle   */
{
  CONFIGURATION init, circle0;

  double sigma;

  printf("\nInput desired smoothness (cm): ");
  sigma = GetReal();
  setSigmaImm(sigma);
  init = defineConfig(0.0, 0.0, 0.0, 0.01);
  circle0 = defineConfig(-100.0, 100.0, -HPI, 0.01);
  setRobotConfigImm(init);
  trackS(circle0);

  printf("\nVelocity is %f", VelocityLinear());
  return;
}

void
user21(void)    /*  CCW circle from outside   */
{

   CONFIGURATION init, circle0;

   double sigma;

   printf("\nInput desired smoothness (cm): ");
   sigma = GetReal();

   setSigmaImm(sigma);

   init = defineConfig(0.0, 0.0, 0.0, 0.0);
```

```
    circle0 = defineConfig(0.0, 50.0, 0.0, 0.01);

    setRobotConfigImm(init);

    track(circle0);

    printf("\nVelocity is %f", VelocityLinear());


return;
}

void
user22(void)    /*  Two circles    */
{
    CONFIGURATION init, circle0, circle1;
    double sigma, radius;

    printf("\nInput desired smoothness (cm): ");
    sigma = GetReal();
    setSigmaImm(sigma);

    printf("\nInput desired radius (cm): ");
    radius = GetReal();

    /* circle to circle   */
    init    = defineConfig(-1.1*radius, radius,  0.0, -1/radius);
    circle0 = defineConfig(-2.1*radius, 0.0, HPI, -1.0/radius);
    circle1 = defineConfig( 2.1*radius, 0.0, HPI,  1.0/radius);
    setRobotConfigImm(init);
    track(circle0);
    track(circle1);
    track(circle0);
    track(circle1);
    trackS(circle0);
    return;
}

void
user23(void) /* Circle Train */
{
  CONFIGURATION init, circle0, circle1, next;
```

```
  double radius;
  int i;
  int numcircles = 1;

  printf("\nInput radius (cm): ");
  radius = GetReal();
  setSigmaImm(0.4*radius);

  init    = circle0 = defineConfig(0.0, 0.0,  0.0, 1.0/radius);
  circle1 = defineConfig( 2.2*radius, 2.0*radius, 0.0,  -1.0/radius);
  next    = defineConfig(4.4*radius, 0.0, 0.0, 0.0);

  setRobotConfigImm(init);
  for (i=0; i <= numcircles; i++)
    {
      track(circle0);
      track(circle1);
      if (i < numcircles)
{
  circle0 = compose(&circle0,&next);
  circle1 = compose(&circle1,&next);
}
    }

  next.Posit.X  = -next.Posit.X;
  for (i=0; i <= numcircles-1; i++)
    {
      circle1 = compose(&circle1,&next);
      track(circle0);
      track(circle1);
      circle0 = compose(&circle0,&next);
    }
  trackS(circle0);
  return;
}

void
user24(void) /* Slalom */
{
  CONFIGURATION init, circle0, circle1, next;
  double radius;
```

```
  printf("\nInput radius (cm): ");
  radius = GetReal();
  setSigmaImm(0.4*radius);

  next = defineConfig(4.4*radius, 0.0, 0.0, 0.0);
  init = defineConfig(-radius, radius, -HPI, 1.0/radius);

  circle0 = defineConfig( 0.0, 0.0, 0.0, 1.0/radius);
  circle1 = defineConfig( 2.2*radius, radius, 0.0, -1.0/radius);
  track(circle0);
  track(circle1);
  circle0 = compose(&circle0, &next);
  circle1 = compose(&circle1, &next);
  track(circle0);
  trackS(circle1);
  return;
}


void
user25(void) /* Obstacle avoidance #1 */
{

  CONFIGURATION init, line, circle;
  double size;

  printf("\nInput radius (cm): ");
  size = GetReal();
  setSigmaImm(0.15*size);
  /*setLinVelImm(20.0);*/

  init = defineConfig(-2.0*size, 0.0, 0.0, 0.0);
  line   =   defineConfig( 2.0*size, 0.0, 0.0, 0.0);
  circle =   defineConfig( 0.0, -0.5*size, 0.0, 2.0/size);
  setRobotConfigImm(init);
  track(line);
  track(circle);
  trackS(line);
  return;
}

void
```

```
user26(void) /* Obstacle avoidance #2 */
{
  CONFIGURATION init, line, circle;
  double size;

  printf("\nInput radius (cm): ");
  size = GetReal();
  setSigmaImm(0.15*size);
  /*setLinVelImm(20.0);*/

  init = defineConfig(-2.5*size, 0.0, 0.0, 0.0);
  line   =   defineConfig( 2.5*size, 0.0, 0.0, 0.0);
  circle =   defineConfig( 0.0, -0.5*size, 0.0, 0.8/size);
  setRobotConfigImm(init);
  track(line);
  track(circle);
  trackS(line);
  return;
}

void
user30(void)
{

   CONFIGURATION init, line0;
   double sigma;

   printf("\nInput desired smoothness (cm): ");
   sigma = GetReal();

   setSigmaImm(sigma);

   init = defineConfig(0.0, 0.0, 0.0, 0.0);

   line0 = defineConfig(200.0, 200.0, HPI, 0.0);

   setRobotConfigImm(init);

   track(init);

   trackS(line0);
```

```
return;
}


/*****************************************************************************
  Function  : user100()
  Purpose   : Displaying the distance obtained by the specified sonar
  Parameters: void
  Returns   : void
  Comments  :
*****************************************************************************/
void user100()
{
  int i, sonar;
  double distance;

  while(1)
    {
      printf("\nInput sonar number ");
      sonar = GetInt();
      EnableSonar(sonar);
      for (i=0; i <300; i++)
{
  distance = Sonar(sonar);
  printf("\nsonar %d distance= %f", sonar, distance);
  waitMS(200);
}
      DisableSonar(sonar);
    }
}
```

APPENDIX J. IMPLEMENTATION DETAILS OF THE WAVELET RECOGNITION SYSTEM

The system code is split into four components: Training Data Processing, Neural Network Training Program, Template Scanning Program and Neural Network Processing Program.

The filename for the x-th sample of Object y's z-th Aspect at angle $w°$ will be: OyAzSxAw.

a. The Training Data Processing Program

The "Training Data Processing" component was implemented in C using the Matrox Imaging Library (MIL) library. It performs the process of reading in images representing a single aspect view of an object in the training database. The filenames of all of the training data is stored in a text file and is the input to this program. The output will be a list of feature vectors corresponding to the interest points detected from analyzing this training data and it will be stored in a lisp format in a file called *trainingl3.lisp*.

1. Read in each training data file.
2. A program is run first to produce the rotated images at the size of 512×512.
3. Transform each aspect & angle image into the wavelet domain.
4. At level 3 in the wavelet domain, detect 10 maximum interest points and a extract feature vector at each point. This information is stored in the output data file called *trainingl3.lisp*. Note that after each set of 10 feature vectors is the identity of the Object, Aspect and Angle classification information that is needed for training.

Bellow, a line of the file *trainingl3.lisp* is shown, containing 50 numbers corresponding to 10 "interest points" and their corresponding feature vectors.

```
(defun train () (main '(
;;Highest_10, level 3, angle 0, figure oc1a0s0
(-17.3000000000000010 -0.6000000000000014 7.7632945599100394
2.7437772789534214 4.8747857976737450 -13.3000000000000010
-0.6000000000000014 2.5363105602360059 1.2928668687180431
1.0453621792006709 -9.3000000000000007 2.3999999999999986
```

```
5.2280813107026365 2.0669463280241538 3.4806936563737985
-3.3000000000000007 0.3999999999999986 2.2464198449293558
0.9445203636985574 0.4874465769380103 -0.3000000000000007
-1.6000000000000014 4.0285701151032116 1.3352789363678150
1.5187340233943245 -0.3000000000000007 1.3999999999999986
5.7299473651630040 2.2450398225978780 2.2525097190972367
4.6999999999999993 -1.6000000000000014 4.5659324623233797
1.6921700892013563 3.0386321169197239 8.6999999999999993
-0.6000000000000014 2.7725534538730385 1.2018148095196759
0.6511144002147382 12.6999999999999990 -0.6000000000000014
10.0000000000000000 3.8939970723089417 13.9707736082173390
17.6999999999999990 1.3999999999999986 4.4644618455921687
1.4596795254323072 1.5890083670703876 ) (...
```

b. The Neural Network Training Program

The Neural Network Training Program as well as the Scanning Program are implemented in Allegro Common Lisp 4.2 on an SGI machine. It takes as input the output file generated by the Training Data Processing Program. This data lists sets of feature vectors and their corresponding Object, Aspect and Angle classification. The output of the Program is the set of NN weights learned.

c. The Template Scanning Program

This program is implemented in C using the MIL library on a PC. The input to this program is a 512×512 image of the scene under consideration. This program converts the image to the wavelet domain and at level 3, performs template scanning. At each template, the top ten interest points are detected and a feature vector for each is calculated. The ten feature vectors for each template is stored in a file called *oyazsxl3.lisp* which is used as input to the next stage of recognition, the Neural Network Processor. Note that the sets of feature vectors for each template of the scene image is stored in this file. They are store in the same format shown for *trainingl3.lisp*.

d. The Neural Network Processing Program

The feature vectors belonging to each template are processed in succession. In the multiple NN case, output from each network is then passed to another lisp routine, that implements the voting procedure. The output from each template is collected into a single output file called *recon.txt*. Below is an example output file. Note that the Location is detected as the center of the template and that the Error provided by the NN is an inverse measure of confidence in the detection of this object.

The example below shows a typical output of the recognition lisp program. The first line explains the output as follows.

- Ai is the number of consecutive windows with equal or close errors (within 10^{-8}).
- Ei is the error.
- Level is the level of the wavelet decomposition.
- Column is the horizontal coordinate of the upper-left corner of the window found.
- Row is the vertical coordinate of the upper-left corner of the window found.
- Width is the horizontal dimension of the window.
- Height is the vertical dimension of the window.
- Classification is the class of the object according with previous convention.

```
  Ai   Ei          level   column    row       width     height    classification

((1 9.398273e-4 (level 3 i_col 10 i_row 8  h_size 36 v_size 28) oc1a0s0a0)
 (2 0.00662132  (level 3 i_col 4  i_row 6  h_size 34 v_size 27) oc1a0s0a0)
 (3 0.019059485 (level 3 i_col 6  i_row 6  h_size 36 v_size 28) oc3a0s0a0)
 (4 0.05333662  (level 3 i_col 26 i_row 28 h_size 38 v_size 16) oc3a0s0a0)
 (5 8.102191e-4 (level 3 i_col 4  i_row 0  h_size 45 v_size 36) oc1a0s0a0)
 (6 8.102191e-4 (level 3 i_col 0  i_row 26 h_size 64 v_size 16) oc1a0s0a0)
 (7)
 (8 0.05333662 (level 3  i_col 24 i_row 28 h_size 38 v_size 16) oc3a0s0a0))
```

LIST OF REFERENCES

[1] Rowe, Neil and Frew, Brian, "Automatic classification of objects in captioned depictive photographs for retrieval." *Intelligent Multimedia Information Retrieval*, MIT Press, Cambridge, MA, 1997.

[2] Grewe, L., Silva Filho, J., and Kanayama, Y., "Recognition using wavelets for the use with a mobile robot explorer." To Appear in *SPIE AeroSense Wavelet Applications II, April 1998.*

[3] Brooks, R.R., and Grewe, L., "On localization of objects in the wavelet domain." *Proc. the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 412–418, July 1997.

[4] Kanayama, Y. and Fahroo, F., "A new line tracking method for nonholonomic vehicles." *Proc. the IEEE International Conference on Robotics and Automation*, pages 2908–2913, April 1997.

[5] Kanayama, Y. and Fahroo, F., "A new line tracking method for nonholonomic vehicles." *Proc. the Fifth IFAC Symposium on Robot Control*, September 1997.

[6] Kanayama, Y., "A path tracking method with neutral switching." Unpublished, 1997.

[7] Castleman, K. R., *Digital Image Processing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1996.

[8] Hornik, Kurt, "Approximation capabilities of multilayer feedforward networks." *Neural Networks*, 4:251–257, 1990.

[9] Kartalopoulos, S. V., *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications.* IEEE Press, New York, NY, 1996.

[10] Ludlow, N., Artificial intelligence for military applications. Class Notes, Department of Computer Science, Naval Postgraduate School, Monterey, CA, 1996.

[11] Rumelhart, D. E., McClelland, J. L. et al., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1 & 2. MIT Press, Cambridge, MA, 1986.

[12] Jain, A. K. and Mao, J., "Artificial neural networks: A tutorial." *Computer Magazine*, 29, March 1996.

[13] Grasp, A., "An introduction to wavelets." *IEEE Computational Science and Engineering*, 2, Summer 1995.

[14] Burrus, C. S. et al., *Introduction to Wavelets and Wavelet Transform: A Primer.* Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

[15] Mallat, S. G., A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Recognition and Machine Intelligency*, pages 674–693, July 1989.

[16] Daubechies, Ingrid. Ten lectures on wavelet. Notes from the 1990 CBMS-NSF Conference on Wavelets and Applications at Lowell, MA, 1992.

[17] Press, W. H. et al., *Numerical Recipes in Fortran.* Cambridge University Press, New York, NY, 1994.

[18] Grewe, L. and Brooks, R.R., "Recognition in the wavelet domain: a survey." Submitted to Computing Surveys ACM, 1997.

[19] Oren, M. et al., "Pedestrian detection using wavelet templates." *IEEE Conference on Computer Vision and Pattern Recognition*, June 1997.

[20] Donoho, D. L., "De-noising by soft-thresholding." *IEEE Transactions on Information Theory*, 3:613–627, May 1995.

[21] Kanayama, Y., Introduction to theoretical robotics. Lecture Notes of the Advanced Robotic Systems Course, Department of Computer Science, Naval Postgraduate School, Monterey, CA, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

4. Chairman, Code CS 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

5. Prof. Yutaka J. Kanayama, Code CS/Ya 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

6. Prof. Gurnam S. Gill, Code EC/Gl 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

7. Prof. Neil C. Rowe, Code CS/Rp 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

8. Prof. Robert McGhee, Code CS/Mz 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5118

9. Prof. Lynne L. Grewe 1
   CSUMB
   100 Campus Center
   Seaside, CA 93955

10. Lt Jader Gomes da Silva Filho.....4
Rua Dona Maria 71/603 Bl. II
Tijuca
Rio de Janeiro, RJ
BRAZIL
CEP: 20541-030

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

DUDLEY KNOX LIBRARY
3 2768 00342804 6